Software projects improvement using PSP: an approach for a R&D center

IÑAKI ETXANIZ
Infotech Unit
Robotiker
Parque Tecnologico, E-48170 Zamudio
SPAIN
http://www.robotiker.com

Abstract: - Extra unofficial hours to maintain the budget, working late to finish on time, project scope renegotiations when we are on the limit, all are things that most software engineers know about to save their projects from failing. When thinking about software process improvement, several methodologies can be found on the literature. In this paper, we will show the application of these tools in a specific case: a R&D centre where, despite the existence of a quality culture and procedures, software projects suffer from the same general illness of over-budgeting and delaying. The method we propose is based in the Personal Software Process (PSP), which differs from others – generally more focused on the company's general structure and organizational issues- in that it deals with the individual work of the developer. Based on previous experiences on the use of PSP and taking into account the particular characteristics of the workplace, a tool is proposed and described.

Key-Words: - Personal Software Process, Improvement, Methodology, Quality, Defects, Size, Estimation.

1 Introduction

On 1994, the Standish Group published its famous "The Chaos Report" [1] where was stated that only 16 percent of the project were successful, while 31 percent fail and 53 percent had serious problems, suffering over-budget and with an average of 189 percent of delay from the scheduled dates. On year 2000, things improved a bit, but still there was a little to be proud of [2] (Figure 1).

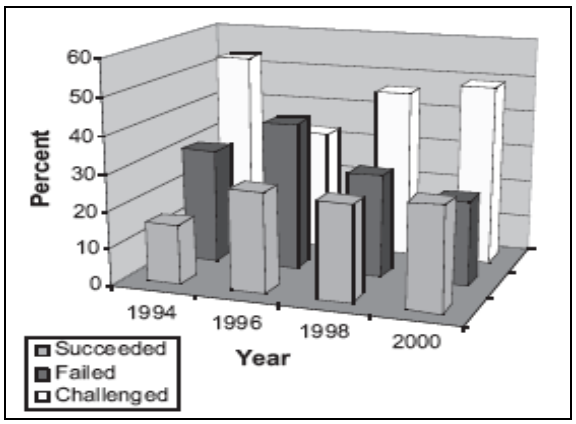

Fig. 1 Project success history, from [2]

Several initiatives have attempt to improve this scene. For example, the Capability Maturity Model (CMM), which was developed by the SEI at the Carnegie Mellon University, or the ISO 9000 standard. The CMM emphasize the software development process control. Some authors dislike it, signaling that this model is like a bureaucratic layer destined to force the developers into rigid institutional environments. An experience with the CMM introduction show that, although a lot of hours where employed defining key areas in the company, the result was a developer manual reflecting only vaguely how things where done in reality, and that was forgotten as fast as it was distributed to the staff.

The Standard ISO 9000 is based in the proposition that if one documents his processes and audit them to correspond with real activities, an environment is created where everyone understand the procedure she has to follow for every activity. Really, in many cases, the documents are maintained only to satisfy the auditors, and keep little relation with the activities the engineers follow day by day.

On the contrary, the PSP focuses on the personal responsibility as mandatory for good working. It is said, also, that this model syntonizes better with our type of culture and with programmers' work -where the individualism prevails over the association- than top-down models like ISO 9000 and CMM [3].

1.1 Problems with the Management

The projects face problems since the beginning. Many of these problems come from outside the team, as too ambitious dates by the management, last hour changes of the scope by the client, or not enough resources. But not all problems can be attributed to the outside; many of them come from inside.

According to the 10 critical success factors of a project [4], several of them are related with the software development process improvement. That is to say, in order of relative importance:

√ Good Planning of the project (factor 3)
√ Correct technical development (factor 6)
√ Precise tracing (factor 8)

Although more focused to big software projects, Humphrey comments [5] that these are intrinsically different from traditional hardware projects, as in the case of construction or engineering. The main difference among them concerns management visibility. In manufacturing, armies or traditional hardware, the managers can walk through the factory, battlefield or laboratory and observe how things go. However, with a team of software development, one cannot tell what they are doing by simply watching.

But the problem is not only to know where the project stands, but in the planning itself. In spite of using several technical tools like PERT or Gantt charts, project managers do not know enough about the work to formulate detailed plans. Although intermediate milestones are clear (finalized specifications, finalized design and similar) usually we do not known clearly when these high level tasks have finished. The requirements work continues during the design, the codification and even into the test, coding habitually starts well before completing the design, etc. That is why plans usually are very generic, as to provide the developers the flexibility to make a creative work. This system would then be the present version equivalent to the one used in the Middle Ages to construct the cathedrals [6], where the developers act like craftsmen, who manage themselves under the guidance of a builder master.

1.2 Problems with the software

A great quantity of errors is introduced in a typical software product. The work to find and fix the errors by means of tests consumes a lot of time -sometimes half of the total development time. But tests only found a part of the mistakes (50 percent is considered a good achievement in the software industry). Some data to remark [7] is:

√ To find and fix a software problem after delivery is up to 100 times more expensive than during the requirements and development phases.
√ Between 40 and 50 percent of the effort is spent in rework.
√ Personal disciplined practices can reduce the defect introduction in a 75 percent.

The PSP helps engineers to detect mistakes early in the process, and since it implies much less effort than to detect them later, the cycle duration decreases and the quality of the final product increases. [8].

The rest of the article is organized as follows: section 2 presents a summary of the PSP. Section 3 introduces the characteristics of the studied work center. Previous experiences related to the use of the PSP are presented in the section 4, whereas in section 5 our PSP-based improvement proposal is described. In the last section we present some conclusions.

2 Overview of the PSP

The Personal Software Process, developed by Watts Humphrey in the Carnegie Mellon University in 1995, has as principal aim the introduction of a certain discipline in the software development process. The model was developed in response to the observation that CMM focused in *what* the organizations should do, but not in *how* to do it. Also, an imputation to CMM was that it is not easily applicable to small organizations. Its design is based in the following principles of planning and quality:

√ Every engineer is different; so, to be more effective, engineers must plan their work and base their plans on their own personal information.
√ To constant improve their work, engineers must use well defined and measured processes.
√ It costs less to find and fix the defects early that later in the process.
√ It is more efficient to anticipate defects that to find them and to fix them.

To understand his personal performance, engineers must measure the time that they spend in every process, the defects they introduce and remove, and the size of the products they produce. Finally, they must analyze the results of every work and use them to improve his personal processes.

2.1 PSP process structure

At the start of the PSP process is the planning phase, having as an input the requirements, which are defined with the help of the client. To calculate size, the engineer first determines the objects needed to construct the product. Then they establish the probable type and the number of methods for every standardized object.

While the engineers do their work (design, code, test…), they record time and defects measures in the corresponding forms.

In the postmortem phase, the engineers actualize their historical data bases. The real size of the program is measured and a summary of the information is joined into the plan-summary form Also, they revise the defects found in compilation and tests, and update their personal review-checking lists to help them to find and fix similar defects in the future. With the personal historical information of size and

productivity, the engineers can better estimate in the future.

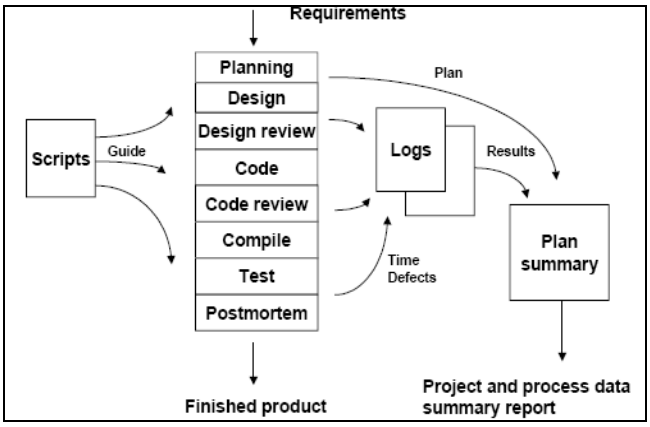

Fig. 2: PSP process flow, from [9]

2.2 PSP measures

During the PSP process fulfillment, three types of measures are carried out: time, defects, and size. All the PSP's metrics are devised form them.

Time is stored in the time form, where you have to annotate the beginning of every task, the ending time, and any interruption interval (a telephonic call, for example). Time metrics are critical to estimate the programming time and calculate the productivity.

Defects are annotated in the corresponding record. Each defect is described by a type (Humphrey defines 10 standard defect types) followed by a textual description, the development phase in which it was introduced, the phase in that was corrected, the time invested correcting it and, finally, it is indicated if it was introduced during the fixing of another defect. Defects can dwell in the code, in designs, or even in the requirements or the rest of documentation.

Whereas the lines of code (LOC) are the habitual measurement of size in PSP, any size measurement that provides a reasonable correlation between the development time and the size of the product can be used. It is important to consider several LOC's categories, like Base, Added, Modified, Eliminated, etc.

2.3 Quality through the PSP

The PSP data demonstrates that even experienced programmers inject a defect into every seven or ten lines of code [10]. The main quality objective in PSP is to find and fix defects before the first compilation. The PSP includes steps as design review and code review, where engineers personally check their work result before it is revised, compiled or tested.

The principle behind this review process is that the developer tends to repeatedly incur in the same mistakes. Therefore, analyzing the information about defects that one has introduced, and constructing a check-list of the actions needed to find these mistakes, the engineers can find and fix defects in a form as efficient as possible.

3 Related work

Hayes and Over realized a study [10] on the performance of 298 participants in a SEI official PSP course. Results showed a substantial improvement in the engineers performance, measured in four dimensions: the size estimation precision, the effort estimation precision, the quality of the product (density of defects) and the quality of the process (early elimination of defects), at the time that it does not significantly concern the productivity.

3.2 Extra effort

As a result of its application in a company of 25 developers [11], it is declared the difficulty that the PSP introduction brings in, on having had to annotate the quantity of interruptions and all the evasions of the normal tasks. Besides, it can be considered a way of spying the developers and measuring his productivity by the management.

Being the PSP a manual process, it requires the engineers to handle a considerable number of forms where to introduce all the significant information on effort, defects, sizes and times (up to 3.300 fields throughout an entire project, if we include all the forms used in the maximum level of PSP's application -level 3.0-). Though the use of tools reduces the needed extra work, it does not eliminate it completely.

3.3 Data Quality

In another study [12] there is demonstrated that the ordinary process in PSP and its manual utilization causes a series of mistakes in the information introduction and analysis. As a result, the conclusions on the improvement that the programmers using the PSP experience can be contaminated by these mistakes, in some way inherent to the methodology.

Among the conclusions, we emphasize the need to rely on an integral tool that supports PSP, which would give the necessary quality to the information and would reduce the required effort to use it.

3.4 LOC and program size

The use of the number of Lines of Code (LOC) to estimate the size of a software application is very widespread, though it has also some detractors. In a study realized at a PSP course, it is suggested that this measure is the cause of many software projects being poorly estimated. It is defined as an intrinsically

ambiguous, vague and inconvenient indicator of the size of the software [13], having sometimes a variation superior to an order of magnitude. Furthermore, for modern programming languages, using buttons and controls in the user interface, nor the physical lines not logical declarations are relevant. [14]
Some alternatives have been proposed, as the Function Point Analysis (FPA), which presents different variants, or the object categorization.
The problem of measuring the size is complicated because in the PSP you have to count not only the final size of the program, but discriminate among original, added, re-used or modified lines, and others.

4 A real industrial environment

The company for which the improvement process has been designed is a technological center, founded in 1985, and that nowadays employs approximately 180 members. Its principal mission is to provide R+D+I (Research, Development and Innovation) services to the companies in the region.
It executes more than 150 R+D projects per year, for more than 130 companies, besides more than 20 internal auto-financed R+D projects. Along its history, the company has taken part in more than 80 European Community founded projects, having approximately 20 active nowadays.
Regarding the software development, almost all the internal units produce some type of software, being one of them - Information Technologies- dedicated mainly to this activity, with a varied application typology: software for embedded systems, for electronic elements, process supervision programs, artificial vision, telecommunications, web applications, software for handhelds (PDAs), etc.

4.1 Quality System

The processes quality has always had great importance in the strategy of the company. It is ISO 9000 certified since 1997 for the software and hardware development processes. The process-based management was established in 2000, and in 2002 the *Silver Q* (a distinction that the regional Basque Government grants to the companies that obtain 4.000+ points in the EFQM-European Foundation for Quality Management model) was obtained.
The projects have to be conducted according to a *Project Management Manual*, which contemplates the steps to generate a project, its planning, control and follow-up, ending and documentation.
To allow this organizational scheme and support the implied procedures, the company has provided itself with some internal tools for quality management.

The first one (*GESIP*) takes charge of the projects' control and follow-up: budget, schedule, work team members, number of hours assigned to each one, planning, and monthly report of hours and resources consumed in the project. It also generates management reports and control diagrams used to monitor commercial activities as well as the economical details (see figure 3).

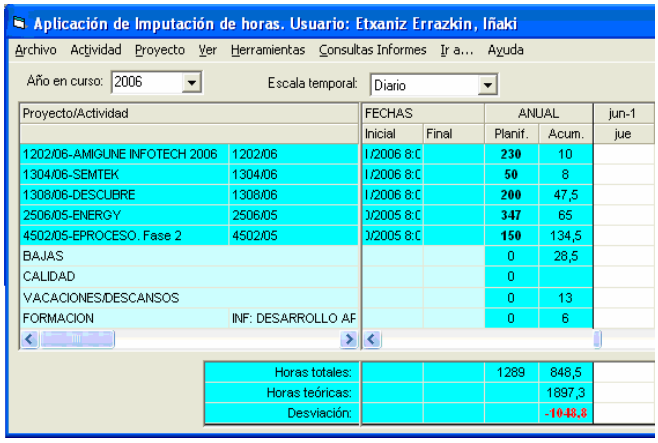

Fig. 3: Hour Imputation Module in the *GESIP* tool.

The second one (*SPRINT*) provides a documentary support to the projects, and a workflow system to organize and automate the day to day work processes.

4.2 Software development standards

The internal *Manual for Software Development Projects* focused to a quality work, defines the software development cycle to be used, contemplates the documents and guides needed for such a project type (Requirements, Design, Tests, etc).
Nevertheless, in spite of this effort in introducing and using a methodology focused on the quality, of having own-developed and external tools that support it, and of having a qualified human staff, a considerable part of the software development projects –and others- continues finishing out of date and over the estimated cost.
Regarding this, we emphasize two reasons for due date delays, from an internal Improvement Group:

- √ Mistakes of planning in the offer (number 5): attributed, among other reasons, to the lack of information on which to base the prediction.
- √ Not programmed tasks (number 10): interruptions of the daily work, unforeseen, visits, urgencies, etc.

5 Proposal

In order to improve the process, we propose here some PSP elements to include, along with an implementation method and the resulting tool. The

data gathered in the PSP methodology can be classified in two categories:

- √ Primary: size, time and defects data, that is independent and previous to any other data; it cannot be obtained from analysis or calculations.
- √ Secondary: information that stems from the primary information. For example, efficiency in the defect fixing, total time spent, etc.

The tool must be capable of gathering so much primary information as possible, whereas the secondary information is automatically calculated and presented to the user in tables or in forms.

5.1 Functionality and integration with the Quality System

It seems clear that the principal objection to the PSP is that it introduces a considerable quantity of extra work, still more if the process is done manually. The fundamental idea is, then, to rely on a tool that should facilitate and automate as much as possible the information gathering and processing, minimizing at the same time the introduced changes in the habitual procedures.

To achieve this, we propose to develop a software module to be attached to the *GESIP* tool, used nowadays for project management, people assignment, resource management, follow-up, etc.

The tool would present menu options to accede to the several data forms to be used in the PSP. The appropriate information fields will then be presented to the user in an electronic manner, to permit first to introduce new data about the project and, second, to consult historical information in order to perform good estimations in new projects.

5.1.1 Time

Concretely, in the hour imputation module the user introduces, nowadays, the hours employed at every task of any project he is working on, with a minimal daily periodicity. To manage the time data input as PSP requires, it would be enough to extend this functionality to allow the introduction of multiple daily inputs in each task. Even better is to use a chronometer as an added functionality. This way the real working time for each task will be calculated, and the interruptions easily filtered.

It is enough to press a button (Start / Pause / Stop) in a graphical interface in order to permit the worker to forget about watching the clock and annotating the exact time. Nevertheless, the time data itself is not enough if it is not tied to the task that the user is working on. This relationship is facilitated, in our case, by the existing tool, which allows in the planning phase to divide a project in tasks and sub-tasks, and afterwards to impute working hours separately to each one.

5.1.2 Size

The size information is more difficult to handle. To begin with, neither the traditionally used LOC, nor the categories of objects, nor the FPA seem to have a sufficient consensus. Nevertheless, size is a basic information when planning the effort needed to perform the project.

Then, we have chosen to use a simple approach: to use only one LOC measure per program -Total LOC- that the user will introduce by hand in the corresponding e-form. No tool is considered for an automatic LOC count, given the heterogeneity of languages and environments used. Besides, the majority of today's development tools include this capability.

For size management, a module would be added so that a final size could be assigned, in the planning phase, to every task. Adding this size up for all the tasks and all the participants would give us the total project size. Additionally, a window would exist in which to define new categories for objects, if desired. Every task might then be defined -*a posteriori* for finished projects, *a priori* for the new ones- in terms of these personal objects.

5.1.3 Defects

For defect management we define an electronic form based on the standard PSP form, which gives the possibility to enter founded defects, to assign them one of the already defined types, the time spent repairing it and the phase of introduction/repair (the latter might be automatically filled, given the planning of the project and the task the user is working on). The phases that PSP uses -requirements, design, codification, etc- are mainly defined in the *Software Development Projects Manual* used in the company, so there is no problem incorporating this information into a form.

Concerning the type of defect, the user will be able to extend or modify the standard PSP categories, attending to the concrete circumstances of his current work. The tool will facilitate the automatic numeration of defects and the link between them, as reflected in the defects form. The historical information will serve to infer the estimated quantity of injected defects of each type in every phase, which PSP uses in a new project planning.

5.1.4 Security and more

The privacy of the user information is also important in order for the tool to be really successful. First, the access to the system needs a private key that

authenticates the user. Second, the PSP information of every user should be kept separated from the information of the others, and only the owner can accede to his data. Nevertheless, to facilitate the group work on the same project, the user is able to "publish" his information anonymously. This would permit group statistics, or defect sharing so that one helps the others to anticipate the most common defects or even facilitate to find the solution of a difficult defect among several coworkers.

A built-in help option gives the user access to the different scripts, procedures and directives that the PSP offers (PSP process script, design-review checking list, code-review checking list, etc). This facilitates the programmer to follow the correct steps in every phase.

6 Conclusions

Although the introduction of process improvement, like the PSP, is not very extended among the software development companies -especially in those of small and medium size- the benefits of its application are evident in the light of the commented experiences. Generally, these methodologies try to systematize the general processes of the companies, without too much repairing in the developer at the end of the chain. The PSP, in turn, affects just to the quality of the individual work of the software engineer.

The principal factors that obstruct the introduction of the PSP are the extra work that supposes the exhaustive information gathering of the methodology; the large amount of forms to be used; and the suspicion that the personal productivity information can be used to some form of evaluation of the workman.

We have analyzed the characteristics of one concrete SME: a center of R&D where software is produced and where, although using already some quality systems, they present the same problems of schedule and costs than in the rest of development companies.

We have defined an implementation of the PSP methodology adapted to the center peculiarity, bearing in mind previous experiences and trying to overcome the difficulties. The principal contribution is the design of a tool to support the PSP, integrated in the quality process existing in the center.

Finally, we expose four areas to work in the near future to develop these ideas.

1. To improve the tool by means of a deeper analysis. Possible new functionalities to be added are: graphical presentations; an additional timer to measure the time used fixing each defect; provide some “intelligence” to the tool so as to warn the user if his productivity is low or he is out of dates.
2. To try the tool in some pilot projects of the company. The results of the experience would be important to decide the introduction of the PSP methodology in the whole center or, otherwise, the modification in those deficient or inadequate aspects.
3. To extend the PSP method with elements of agile development technologies, like XP, Scrum, etc.

References:

[1] The Standish Group International, Inc, The Chaos Report, 1994

[2] The Standish Group International, Inc, Extreme Chaos, 2001

[3] Keuffel, Warren, Coding Cowboys and Software Processes, *Crosstalk, the Journal of Defense Software Engineering*, August 1997

[4] Pinto, J.K., Slevin, D.P., Critical Success Factors in Successful Projects Implementation, *IEEE Transactions on Engineering Management*, Vol. 34, No.1, 1977

[5] Humphrey, Watts S., Why Big Software Projects Fail: The 12 Key Questions, *Crosstalk, the Journal of Defense Software Engineering*, March 2005

[6] Raymond, Eric S., *The Cathedral and the Bazaar*, Cambridge, MA: O’Reilly Publishers, 1999

[7] Boehm, B., Basili, V.R., Software Defect Reduction Top 10 List, *IEEE Computer*, January 2001

[8] SEI, Carnegie Mellon University, Pursue Better Software, Not Absolution for Defective Products, *news@sei interactive*, March 2001

[9] Humphrey, Watts S., *Introduction to the Personal Software Process*, SEI Series in Software Engineering. Addison-Wesley, 1987

[10] Hayes, Will and Over, James W., The Personal Software Process (PSP): An Empirical study of the Impact of PSP on Individual Engineers, Technical report, *CMU/SEI-97-TR-001*, December 1997

[11] Hemdal, J. Erik and Galen, Robert L., PSP-A few unexpected lessons, *RTP-SPIN Meeting Presentation*, Feb 2000

[12] Disney A., Johnson P., Investigating Data Quality Problems in the PSP (Experience Paper), *Sixth International Symposium on the Foundations of Software Engineering (SIGSOFT'98), Orlando, FL.*, November, 1998

[13] Schofield, Joe, The Statistically Unreliable Nature of Lines of Code, *Crosstalk, the Journal of Defense Software Engineering*, April 2005

[14] Jones, C., *Software Quality*, International Thomson Computer Press, 1997: 333