Towards a Flexible Tool for Supporting Data Collection & Analysis in Personal Software Process (PSP)

MOHD HAIRUL NIZAM MD NASIR, AZAH ANIR NORMAN, NOOR HAFIZAH HASSAN
Faculty of Computer Science and Information Technology
University of Malaya 50603 Kuala Lumpur
MALAYSIA.
hairulnizam@um.edu.my , azahnorman@um.edu.my

Abstract: - Personal Software Process (PSP) ultimately provides software engineers an excellent framework and practice that can help them to improve the quality of their work, by analyzing their performance statistically and helping them to achieve realistic goals set by them. Besides, PSP offers many benefits to software engineers. However, through findings and studies, it is found that the Personal Software Process adoption problem may be caused by the overhead in data collection, manual execution in data analysis, and inflexibility of process definition. This paper presents in details the factors that influence the PSP adoption problem and explains the need for automated tool to support the adoption of PSP. It is believed that with the highly flexible automated tool support, it can give the flexibility to the software engineers to manage their process definition rather than staying freeze. Other than that, it can minimize the overhead during the data collection and data analysis phases. Software engineers should be easily monitors, measure and improve their software development process by using other additional features provided by this tool.

Key-Words: - Personal Software Process, PSP, Automated Tool, Software Process, Flexible Tool.

1 Introduction

'Software development is a challenging undertaking that is often critical to the safety of humans and the welfare of businesses' described by Zahran in [17]. Since more than forty years ago, the software development experience has not succeeded in overcoming this problem. Around 1960's, the term "Software Crisis" emerged to describe the software industry's inability to provide customers with high quality products within schedule and budget. Between 1985 and 1987, two people died and four others were seriously injured after they received massive radiation beamed via Therac-25 radiation therapy machine. Successful investigations revealed that defective software is among the various factors leading to this accident Leveson *et*. al [10]. Another example is the delay for over 16 months of the opening of Denver International airport, and over 100 million dollars exceeding the budget in the construction cost reported by Swartz in [15].

As a result, inspired by the efforts of Deming [1] and Juran [9].There has been an increase in attention and focus on the discipline of software engineering. New software engineering methods, techniques and tools have been developed to gain more predictable quality improvement results. They are needed to manage the complexity inherent in large software system. Many process standards on organizational level such as Software Process Improvement and Capability dEtermination (SPICE), International Standards Organization (ISO) 9000 for project by Siyal such as in [19] , Capability Maturity Model (CMM) by Paulk [11], Capability Maturity Model Integration (CMM-I), BOOTSRAP; Team Software Process (TSP) by Humphrey in [8] and eXtreme Programming (XP) which are deals on team level; and on individual level namely Personal Software Process (PSP) by Humphrey [5]. All these standard models have been proposed to assist organizations, teams and individuals to achieve results that are more predictable by incorporating these proven standards and framework into their software development process. Therefore, controlling and improving the processes used to develop software have been proposed as a primary remedy to these problems.

This paper specifically will discuss in details the PSP adoption problems in current software engineering practices. It also explains the need of an automated tool in order to solve these identified problems. The automated tool must incorporate 3 features to support the PSP framework in software development namely minimized data collection, automate execution in data analysis, and provide flexibility in process definition.

This paper is divided into 5 sections. The first section introduced the current situation of software engineering disciplined, the importance of having standard software process improvement model and gave examples of existing software process standards that are used on an organizational, team and individual level. The second section gives a brief overview of the PSP, while the third section discusses the problems that influence the adoption of the PSP. The fourth section discusses the importance of having an automated tool and presents the features that the automated tool must have. The last section summarizes the main points of this paper.

2 Overview of the PSP

The concept of the PSP by Humphrey, originated from Watts Humphrey of the SEI as a response to the observation that the CMM was not applicable to small organizations [6]. Humphrey developed a software development process similar to CMM level 5 for the smallest possible (individual) organization.

The PSP is a self-improvement framework that includes defined operations, measurement and analysis techniques to assist software engineers to understand and build their own skills in order to improve their own personal performance. The purpose of the PSP is to help software engineers to learn and practice those software methods that are most effective for them. Each new program written can give benefit from the collection of the data of the past projects, and provide new insights to improve planning, productivity, and quality for future work.

However, software engineers must adhere to a thorough and complex process to make this improvement happen. As illustrated in Figure 1, there are 7 versions of PSP processes begins at Level 0.0 and progresses in their process maturity up to Level 3.0 which involves 76 documents consisting of forms, process scripts, and instructions packaged together in PSP to be used by software engineers to capture performance data. It is implemented in seven incremental levels. The next level consists of all the methods in the current level plus one or more new features.

When a software engineer utilizes the PSP, he or she initiates it by doing planning task with a set of predefined worksheet type forms. Then, while moving through the design, code, compile, and test phases, he or she uses other forms to maintain detailed records regarding the time used in each development phase as well as defects injected into and removed from the phase. As soon as the project is completed, there is a final "postmortem" phase in which the software engineer analyzes all the data gathered about the project and calculates values such as *Lines of Code (LOC) per Hour* and *Defects per Thousand Lines of Code (KLOC)*. He or she computes such values for the current project, and then calculates them again as "to date" values, not only for the current project, but also for the entire set of similar projects used in planning. Therefore, at the most basic level of measurement framework, the PSP involves two main activities, which are collecting primary data such as size, defect, and time measures, and analyzing this data to produce derived or to-date measures.

Growing evidence shows that the PSP works. Many researchers have performed studies on the use and payback of applying the PSP in education and industrial setting. Some studies show great improvement in the quality and productivity of the software developed, while other studies report low acceptance of the PSP and some show no improvement at all. Hayes & Over performed an empirical study in [4] extensively with 298 engineers who spent more than 15,000 hours writing over 300,000 lines of code and removing about 22,000 defects to examine the effectiveness of the PSP on the performance of engineers. The results of the study provide very impressive evidence to support the effectiveness of the PSP. Over the projects completed, 50% of the engineers reduced their size estimation error by a factor of 2.5. Product quality, which is measured by defects found in the product at unit test, improves 2.5 times. Other than that, the median in time estimation and effort estimation improve 1.75 times higher and median in overall defect density was reduced by a factor of 1.5. It is notable that PSP improves the performance in the first four of these dimensions without any loss in productivity. Another study conducted by Watts Humphrey reported the results in [7] of 104 engineers taking the PSP course. After analyzing the data collected from his students, he stated that two objectives had been achieved namely productivity and quality, and estimation accuracy. It is reported that the test defects were reduced by 73.2 percent while defects density fell gradually from an average of 116.4 defects per thousand LOC for assignment 1 to 48.9 defects per KLOC for assignment 10. Besides, the estimation accuracy of the engineers was also increased, where for assignment 1, the engineers' estimation accuracy average is about 32.7 percent whereas for assignment 10, the estimation accuracy increased to 49.0 percent within 20 percent of their actual time. These results show that the engineers who took the course improved

better in estimating the amount of time in completing their assignments.

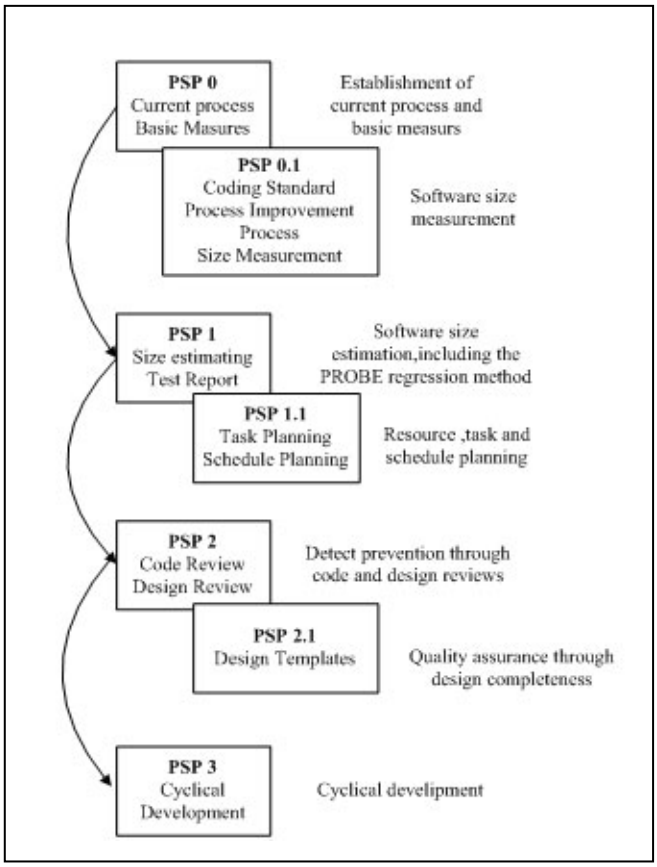

Fig. 1: The PSP Process Evolution

However, the PSP is still new, and undoubtedly, it has not been proven to be suitable for everyone. A study conducted by Shostak [13] against 28 software engineers at CAE Electronic Ltd. who were provided with the PSP knowledge and training from McGill University, reports poor adoption of PSP in industry settings. During the study, the researchers found that seven months after the end of the training, only 46.5% of the engineers kept using the concepts of the PSP. Another study performed by Disney reported that the quality issue of the data recorded during data collection and analysis phases [3]. In her study, she discovered 1539 primary errors and also 90 errors that indicated deeper problems in the collection of primary data measurement. She was also found that incorrect data recorded will lead to incorrect analysis of said data where it is not only cause incorrect values in the current project but may ripple through to future projects.

3 Adoption Problem in PSP

Several studies have shown that the PSP appears to help in improving software development, but is not the complete solution to the software development issue. Originally, PSP has several disadvantages that may discourage software engineers from using it as a framework to improve their performance. This section attempts to clarify the factors that influence the adoption of the PSP in software engineering community.

3.1 Overhead in Data Collection and Analysis

Deploying the PSP to put into practice has significant effects in terms of cost and degree of commitment. Through literature findings, PSP is an empirically based process improvement framework focusing on individual software engineer, which requires him or her to create or print out forms. All significant measures regarding log effort, defects during software development, size of the software and other measures, are manually recorded. This approach collects data about the work product and processes.

As many efforts is put into producing a high quality software product, most software engineers find it troublesome to manually record defect data into the printed log form. In addition, manual PSP requires them to use stopwatch to record the interruption or defect removal time. This manual data collection significantly increases the engineers' workload. The forms will be used as an information reference to sustain project estimation and quality assurance. After many projects, each engineer accumulates a large paper database of their historical data. As a result, this approach creates extensive overhead and extra workload due to form filing. Other than that, a large number of different documents and forms need to be managed and organized properly, so that they are easily accessible when required.

Each PSP level introduces new measures to help engineers to manage and improve their performance. As a result, the input field required increases from each level to each level in PSP. Consider a situation where a software engineer has reached PSP3.0 level. Even though the software development project might be small, it has 7 types of PSP documents and forms that are equivalent to 36 pages, along with hundreds of fields to be filled. It becomes even harder when all the forms have correlations with each other. PSP is termed as time consuming in term of data collection and analyzing stage. In data collection for example, there are many PSP forms and each form has many fields that

require the engineers to fill in. In data analysis process, PSP needs the forms to calculate and analyze recorded data in order to gauge the engineers' performance against software development work. As shown in Table 1, when the software engineers are in PSP3.0 level, they are required to fill in approximately 1115 fields. Assuming one field takes an average of 15 seconds to fill-in (including overhead in context-switching between actual work and recording work, doing calculation and analysis), a total of 278.8 minutes or 4.65 hours is required for the engineers to completely fill-in all the fields. This problem is also supported obviously by Humphrey statement where 'When the project is completed, it can take up to an hour to gather all the data together and calculate performance metrics' [5].

The calculation of measures for the PSP, as well as the process of collecting data, regardless of effort and defective data, both are repetitive and tedious. For instance, defect log data are requested in both Defect Recording Log form and in the Project Plan Summary. This means overlapping activities occur and redundant data exist in different documents within the same software project. If these overlapping activities and redundant data are avoided, the time spent in the recording stage can be minimized.

Modifying data in the PSP requires engineers to stick to the PSP forms. The forms in the PSP are very tightly tied to the PSP process and are always associated with a project and a phase in the development process. For instance, an engineer might record "1.00 pm to 4.00 pm, working on project A" in the testing phase. It seems simple and trouble-free theoretically, but in practice, it is very important to define unique projects for every development activity, determine the phases to be assigned, and record individual entries each time the software engineer switches to another different task or project. This is a cumbersome task.

Problems also occur when changes need to be made to an entry or process. If the changes are small or minor such as renaming the phases, it will not be that difficult. However, for bigger or major changes such as changing the size measurement or estimation technique, it will be very complex as it may affect many forms and scripts. This is because the PSP data are dependent on each other, and the forms are connected to each other. Software engineers must place the PSP forms near them, and then refer to various values for at least 45 times in PSP2.0 level, this without taking into account the size and time estimation.

Adding an additional measure to software engineers' work product and processes may cause a huge psychological overhead to the engineers, and thus, the possibility of making errors during data collection and analysis are high. This is proven by Disney *et al.* (1998), where significant data quality issues with manual PSP were found in [2]. In such a case, not all defects were recorded because the overhead in recording was too expensive. Furthermore, higher overhead in recording give a bad influence against the quality of PSP data. Without tool support, the quality of the data collected tends to be low.

Many complain that PSP is psychologically disruptive, because it requires the engineers to stop for a second to record the data such as defects or effort log on the PSP form, and requires them to "context-switch" between software development work and process recording. It is recommended that the engineers keep a stopwatch by their desk to keep track of all interruptions and effort log. Both stopping and context-switch process disrupt the flow state of software development work. In addition, to be accurate, the PSP requires the engineers to record "idle time". As a result, every interruption such as phone calls generates an additional recording activity.

The overhead of PSP-style metrics collection and analysis is one of the major issues in PSP adoption. In this case, four significant drawbacks regarding this issue were identified, which are:

3.1.1. Low in manageability and hard to organize

After many projects, an engineer accumulates a large paper database of his or her historical data. As a result, this approach creates extensive overhead due to form filing and the number of different documents. It is hard for the forms be managed and organized.

3.1.2. Humans are naturally error prone

Adding an additional measure to an engineer's work product and processes may cause a huge psychological overhead for the engineer. Unconsciously, when there are hundreds of fields and forms to fill-in along with extensive overhead in the data analysis, the possibilities of making error during data collection and analysis are high. In addition, it becomes harder when all forms have correlation with each other. This is proven by Disney *et* al, which show higher overhead leads to data collection error [2].

3.1.3 Manual execution is tedious and increases workloads

Practically, engineers are trained to produce high quality software. If the improvement process model increases their workload, the adoption possibilities tend to be low. The need for a fully-automated tool can combat human assistant. It is paperless, and thus, the workload on managing the correlated data from the different projects is reduced.

3.1.4 Time Consuming and Psychologically Disruptive

There are many PSP forms and each form has hundreds of fields that require the engineers to fill in, thus consuming the engineers' time. Besides, PSP is psychologically disruptive, where each time the engineers discover a defect, they must stop their work and record the defect in their Defect Recording Log. As a consequence to this, the engineers will refuse to employ this discipline into their work since it is better for them to fully-utilize their time in developing and enhancing their programming work.

While measurement in PSP has many benefits, it introduces overhead to the users. The overhead of manually analyzing and recording these personal measures by hand will outweigh the benefits of the process. The improvement process model employed should not burden the engineers' shoulders. Engineers are not supposed to worry about the improvement effort while they are in the midst of developing software.

3.2 Freezing of Process Definition

'Originally, the PSP provides the basic phases for software development' [5]. Starting from the planning phase until the postmortem phase, PSP covers the basic tasks for developing a software product. This software model tailors the waterfall software life cycle model as illustrated in Figure 2. In PSP1, the Test Report Template is introduced, where it offers software engineer the control to determine test cases in the development phases. When it is time to execute the testing, the test cases are prepared to run. This development process can support the V-shaped software development model, where the test cases are developed as early as the design phase. At the same time, PSP3.0 can deal with the spiral life cycle model.

Although these approaches are generic, it does not allow software engineers to move back and forth between phases. Rather, software engineers have to move sequentially through each phase that is appropriate for the process. For instance, when we are in the coding phase and then discover a plan defect, we cannot go back to the planning phase to fix the defect. In Disney thesis[3], she had classified this case as a sequence error. 'The Time Recording Log shows a student moving back and forth between phases such as Compile and Test phases, instead of sequentially moving through the phases appropriate for the process' [2].

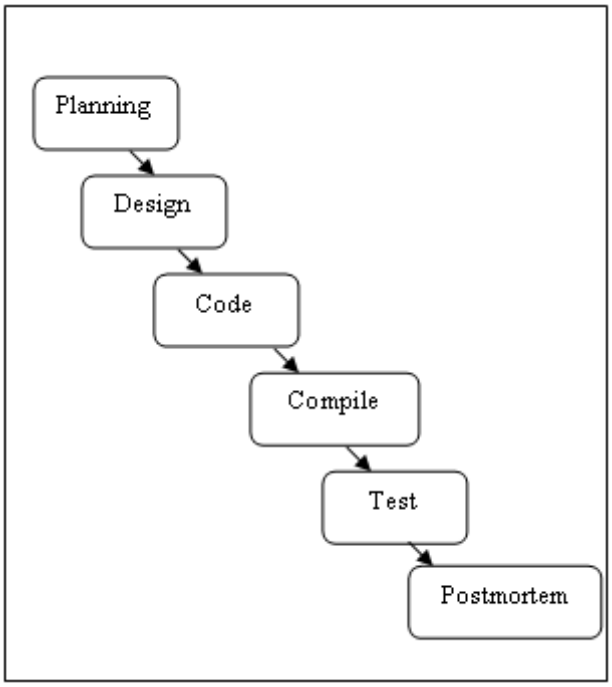

Fig. 2: Software Process Model of PSP

This limitation enforces process restrictions on software engineers, thus making the process improvement too impractical and inflexible. Moreover, in practice, most contemporary and large scale software development project cannot be done using the waterfall or V-shaped life cycle model, but it is more focused on risk oriented or object oriented life cycle. As mentioned by Lin et al.,[18] 'risk is the traditional manner of expressing uncertainty in the systems life cycle' .In a development environment like IBM's Visual Age for Java, the code is automatically compiled as it is saved, thus, makes the compiling phase needless.

There are diverse development process models used by software engineers such as rapid prototyping, evolutionary development model, incremental development, spiral model and others. On the basis of their prior experiences, software engineers tend to utilize the development process that is tailored and well-suited to their personal needs and the software development situation, as they know what

adjustments they need to make and when to make them. The software engineering community has always emphasized the quality of software design as primary for the effective achievement of software development projects [20]. Consider a situation where a software engineer does some planning work before design. During his planning process, he develops a framework for his software development project and writes down the significant elements needed, the scheduling and the estimates of resources and effort required for his project. Once completed, he then shifts to the design phase. In the design phase, after thorough observation and analysis, he makes several changes that affect the framework he developed during the planning phase. As a result, he has to backtrack to the previous phase and make the necessary changes. In this simple scenario, the engineer does not tailor to the PSP waterfall model. In his process, he prefers to use his own creativity to assist him in developing a better plan.

It should also be noted that the PSP is actually more on code-centric development activities. It is geared specifically towards the development of software, whereby software engineers need to follow the process flow sequentially, starting from planning, designing, coding, compiling and finally testing the system, as illustrated in Figure 2. However, software engineers do more than just coding or programming. PSP makes an assumption that software engineers understand well the user requirements. But eliciting requirements is a key task that can be considered as a phase in PSP. Moreover, an individual software engineer may wish to add a phase specifically for the non-production stages such as risk management, requirements negotiation and formal specification. They are also required to perform non-programming activities such as system maintenance, writing reports, preparing technical presentations and documentation. If a software engineer is asked to prepare technical documentation in relation to his programming work, how can he measure this since it is not covered in the PSP waterfall model? Besides, the PSP splits the software development project into process levels only. In actuality, maybe a software engineer favours splitting his project into processes and sub-processes instead, to make it easier to monitor. But with manual PSP, if there are many processes along with sub-processes, this method becomes tedious, complex and difficult to manage. Whatever forms or fields that can be customized, should be identified first since the engineers as the users, are capable of doing such things as defining or customizing new processes, adding new phase or deleting a defined process.

Therefore, the improvement process model should not freeze but rather, support other tasks that engineers do, so that it can be applied to many other different processes and can be widely adopted by the software engineering community. However, since Humphrey intended that engineers should modify the PSP for their own needs, the PSP can be adjusted to fit any of the software development models mentioned above [6]. ‘The PSP of its methods should be adjusted to one's own technology, practice, strengths, and weaknesses’ [6].

4 The Need for an Automated Tool Support

While the measurements required by the PSP have many benefits, they introduce overhead to the users. The overhead of recording these personal measures by hand and manually performing the analysis outweighs the benefits of the process. Any improvement process model employed should not impose any added burden on software engineers. Software engineers are not supposed to worry about their improvement efforts while they are in the midst of developing software.

In addition, many software engineers have an automation-oriented mindset and to them, it is very time consuming to have to look at all the unnecessary documents during the software development process. Unnecessary documents here refer to those documents that are not directly related to the software development project that they are currently undertaking. Overhead, either in the collection or analysis of data can be reduced through tool support that makes manual recording of time, defects, and size of program effortless and more accurate. Therefore, a viable solution concerning this issue is to provide an automated tool to support the PSP. This is required in order to simplify the entire task involved in the PSP by relocating human effort into an automated tool. This makes data more convenient and easier to organize, achieving a paperless working environment.

The absence of an automated tool is one of the major difficulties in continuing to continue the usage of this discipline. Thus, it is recognized that automated support would provide time saving during development. In the finishing stage of a project, where performance data is calculated and posted, time saving would be more noticeable. However, if an automated tool is poorly designed, it

can lead to a waste of time and effort in using it, as compared to doing the PSP manually. This section looks at how the automated tool can be improved by incorporating the properties mentioned below.

4.1 Towards Minimizing Data Collection and Analysis

An automated tool should reduce the overhead of software engineers' improvement by automating as much data collection processes and analysis calculations and conversions as possible. If engineers do not use automated tools, this means they need to do a lot of paperwork.

Although PSP requires software engineers to fill-in hundreds of data fields, in actuality, the PSP data consists of two types; primary data and secondary data. Size, time and defects data are considered as primary data. This type of data is independent and cannot be derived from any calculations or analysis against prior data; the only way to obtain this data is through the collection process. The PSP automated tool should be able to do as much collection of primary data as possible and this can be done by providing electronic data forms which can be easily accessed on demand. Besides that, whenever possible, the tool should also be able to collect data automatically such as timestamp data, so that users are less likely to record these data, thus avoid having incomplete and inaccurate data of their work. For example, users should not have to fill-in time log entry fields but instead, the users have the power to override or correct any of this automatically collected data. In this case, no stopwatch is needed. This way, timing data can be made accurate to the second. All these personal measures should be stored conveniently for later analysis.

Secondary data, on the other hand, refers to data that are derived from primary data. Secondary data, such as defect removal efficiency and to-date percentage, are dependent on and are derived from prior project data and these data are also known as "carry-forward values". Some secondary data are also derived from performing mathematical calculations and analysis. Besides, there are larger amounts of secondary data compared to primary data with an average ratio of 4.20 as illustrated in Table 2. The PSP tool should be able to automatically calculate secondary data, and display and insert them into the electronic PSP forms. At this point, if the calculation and analysis processes are done automatically, the time spent in both processes can be minimized, thus reducing the analysis overhead. The core function or requirement of the PSP automated tool is the ability to reliably automate all of the PSP calculations. In addition, the tool should also be capable of performing inter-project management by calculating and displaying all "carry-forward values" from a prior project to the current one. As shown in Table 2 in Appendix B and Figure 3, if software engineers using an automated tool to support their data collection and analysis, the total times spent will reduced by a factor of 3.31,4.24,4.50,4.85,5.15,6.40 and 7.97 for PSP level 0,0.1,1,1.1,2,2.1 and 3.0 respectively. Obviously, it shows a great improvement in terms of time usage.

Naturally, human beings will make mistakes in any practice especially when there are a lot of pressures in doing their work. Therefore, one of the possible solutions to minimize data errors is by employing improved automated tool. Automating much of the data entry, analysis and transfer will reduce the opportunity to make mistakes. This improved automated tool will capture and analyze the significant errors as presented in more detailed in Disney *et* al. [2] thesis on the subject of data quality in PSP. It will also attempt to eliminate many of the data errors that Disney found. Also, data consistency is ensured, whereby engineers do not have to copy the data by hand. Consider a scenario when changes to one's personal process would require changes to the forms. The forms are stored electronically and printed out as needed. If the forms would have to be changed, changes are made and then, they are saved to disk. However, a correction applied across a project can take time, forcing the recalculation of many fields. The scenario above is also an example of what is meant by automated analysis supported by this tool.

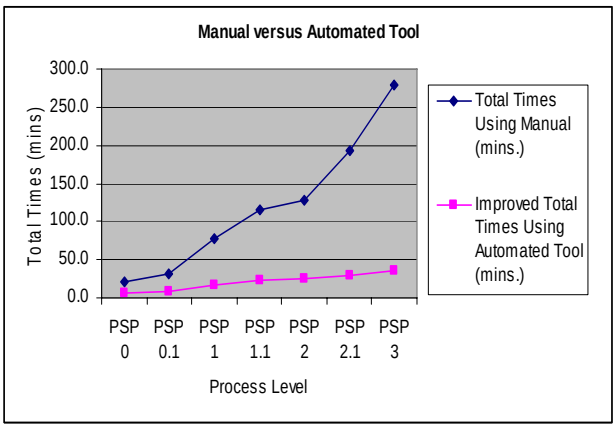

Fig. 3: Comparison of Total Times between Manual and Automated Tool Support

4.2 High in Flexibility

'A phase is defined as a structured task of the software process' [5]. 'A process consists of several phases with each phase having a unique name. The PSP phases are structured in an order that is natural in the development process. However, the tool should be structured such that the user can run in any phase he feels comfortable with' [14].

The freezing of process definition problem can be solved if software engineers have full control in determining their own development process. The automated PSP tool should allow for phase customization, in which it allows software engineers to define their own development process and to follow it. In addition, the tool should not enforce a specific development process model on software engineers, like the PSP does. It should offer the software engineers full control and the ability to utilize their own creativity in order to determine their own way of working. If software engineers do not desire to have a plan phase or to follow a sequence like the PSP waterfall model in their software development project, the tool should support this desire. Software engineers should have the privilege to define and customize their own defined process, with the PSP tool continuously supporting data collection and analysis, based on their process definition model. In addition, the tool should allow software engineers to incorporate other non-programming activities such as system maintenance, report writing, or documenting with the PSP. However, if the engineers desire to employ similar process as the PSP waterfall model, they may still do so. The aim is to provide an alternative in order to achieve flexibility through user customization.

Figure 4 illustrates the flow chart of the proposed automated PSP tool. To secure process data, the tool requires a valid personal password that is associated with a username for each user to access the data. Since each user is unique, each user's data can be distinguished. For flexibility, users can either use standard PSP or use their own process definition. In addition, the users are able to create and customize new process definitions. It also allows users to add new sub-processes in their process definition. The new process definition is then saved in the personal database once the users have completed the process. The tool will then generate the PSP forms and documents, and continuously support data collection and analysis based on the users' process definition model.

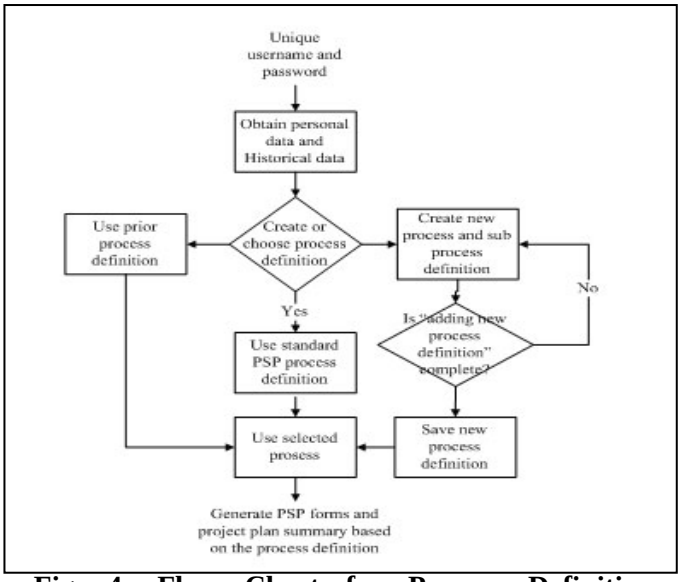

Fig. 4: Flow Chart for Process Definition Customization

5 Conclusions

This paper highlights on the need of a software tool support for a personal software process. The basic notions and concepts were extracted from the Software Engineering Institute, which introduced Personal Software Process (PSP) concept and Capability Maturity Model (CMM) [14].

Although the philosophies and disciplines associated with software engineering at an individual level have found an expression through the PSP, it was apparent from findings that there should be a fully automated tool to support this manual framework. In this context, fully automated means that the tool automates the entire process and level of the PSP including collecting and analyzing data, managing personal measures, and guaranteeing data integrity and consistency along with providing access to the PSP forms. The core requirements of the PSP automated tool are to facilitate the collection of personal measures as a complement to the development process, to put forward the data collected for analysis, and to provide software engineers with important process feedbacks and reports.

Thus two features have been proposed that the automated tool should have. Firstly, as the core function, the automated tools must reliably support data collection, and computation and analysis of measures which will lead to an increase in the adoption of personal software process improvement. Primary data can be collected using electronic forms provided by the tool and some inserted

automatically, while secondary data can be automatically computed. Duplicate data need only be entered once, and will automatically be displayed on the PSP forms when needed. This can reduce overhead and time spent during the recording and analysis stage.

Secondly, as an addition to the PSP tool being fully automated, flexibility in process definition should also be taken into consideration. The automated tool should give the user flexibility in process definition. To achieve this, the tool must allow software engineers to define their own development process and follow it. The automated tool should not enforce a specific development process model on the engineers. Nevertheless, if the software engineers want to employ existing process models such as the PSP waterfall model, they may do so. Regardless, the software engineers should be able to define and customize their own processes, with the tool continuously supporting data collection and analysis based upon the defined processes.

The benefits of employing the improvement process should outweigh its costs. A software engineer should be able to see the reimbursement of his improvement efforts as soon as possible after using the PSP automated tool. However, the notion and the principles behind the PSP must be first understood; the tool is just a way of providing a more convenient working environment for software engineers.

References:

[1] Deming, W.E., *Out of Crisis*, MIT Center for Advanced Engineering Study. Cambridge, MA.1982

[2] Disney, A. and Johnson, P., Investigating Data Quality Problems in the PSP. *Proceedings of the ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering*, Vol. 12, No.6.1998, pp.143-152.

[3] Disney, A. M. *Data Quality Problems in the Personal Software Process.* M.Sc. Thesis, University of Hawaii. 1998.

[4] Hayes, W. and Over, J.W. The Personal Software Process: An Empirical Study on the Impact of PSP on Individual Engineers**,** *Technical Report CMU/SEI-97-TR-001*. Software Engineering Institute. Pittsburgh, PA: Carnegie Mellon University. 1998.

[5] Humphrey, W. S., *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley, 1995.

[6] Humphrey, W. S., The Personal Process in Software Engineering**,** *Proceedings of the Third International Conference on the Software Process, Reston, VA*, pp. 69-77.

[7] Humphrey, W. S., Using a Defined and Measured Personal Software Process, *IEEE Software.* Vol. 13, No. 5, 1996, pp. 77-88.

[8] Humphrey, W. S. *Introduction to the Team Software Process*. Reading, MA: Addison-Wesley, 2000.

[9] Juran, J. M., Gryna, F. M., *Juran's Quality Control Handbook*, Fourth ed. New York, New York: McGraw-Hill Book Company, 1998.

[10] Leveson, N.G. and Turner, C.S. An Investigation of the Therac-25 Accidents, *IEEE Computer*, 1993.

[11] Paulk, M. C., The Evolution of SEI's Capability Maturity Model for Software. Software Engineering Institute, *CMM Evolution*. 1994.

[12] Paulk, M.C., Chrissis, B.M. and Weber, C., Capability Maturity Model for Software Version 1.1., *Technical Report CMU/SEI-93-TR-24. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.* 1993.

[13] Shostak, B. Adapting the Personal Software Process to industry *Software Process Newsletter #5.* 1996.

[14] Software Engineering Institute (1998). *Why Use the PSP – Overview*. Pittsburgh, PA: Carnegie Mellon University. URL : http://www.sei.cmu.edu/activities/psp/WhyPSP.htm

[15] Swartz, A.J., Airport 95: Automated baggage system? , *ACM Software Engineering Notes*, Vol. 21, No. 2, pp. 79-83, 1996.

[16] Webb, D. and Humphrey, W., Using the TSP on the Task View Project, Crosstalk 12. 1999.

[17] Zahran, S., Software *Process Improvement: Practical Guidelines for Business Success. Reading*, Addison-Wesley Professional 1997.

[18] Lin, L., Lee H.M., Lee S.Y., Lee, T.Y. *Fuzzy Evaluation for the Rate of Aggregative Risk in Software Development***,** Proceedings of the 6th WSEAS International Conference on Applied Computer Science. 2007. pp.43-47.

[19] Siyal, M.Y., A Novel Framework for Business to Consumer E-Commerce.*2nd WSEAS International Conference on E-Activities.* 2003.

[20] Fernandes, S.M., ,Belix J. E., Melnikoff, S.S.,Spina, E., Confronting Antagonistic Views of Software Design, 4th WSEAS International Conference on Applied Mathematics and Computer Science. 2003. pp. 223-227.

APPENDIX A

Process Level	Approximate Fields						Total Fields	Total Time (minutes)
	* Header	** Log Form	*** Summary	**** PIP	**** Templates	****** Checklist		
PSP0	14	12	60	0	0	0	86	21.5
PSP0.1	23	12	79	13	0	0	127	31.8
PSP1.0	52	12	89	13	145	0	311	77.8
PSP1.1	76	12	100	13	145	116	462	115.5
PSP2.0	88	12	170	13	145	147	515	128.8
PSP2.1	140	24	197	13	247	147	768	192.0
PSP3.0	198	50	460	13	247	147	1115	278.8
Total							**3384**	

Table 1: Number of Fields for Each PSP Level

* Header including Name, Date, Program, Program Number, Instructor and Language field in each of PSP form

** Log Form including Time Recording Log, Defect Recording Log and Issue Tracking Log forms

*** Summary including Project Plan Summary and Cycle Summary forms

**** PIP is known as Personal Improvement Proposal form

***** Templates including Test Report, Task Planning, Schedule Planning, Operational Scenario, Functional Specification, State Specification and Logic Specification,

******Checklist including Code Review Checklist and Design Review Checklist forms

APPENDIX B

Process Level	Primary Data Approx. Fields	Secondary Data Approx. Fields		Ratio Secondary Per Primary	Total Times Using Manual (minutes)	Improved Total Times Using Automated Tool (minutes)	Improved Factor (Times)
		Derived From Calculation	Carry Forward Values				
PSP 0.0	26	60	0	2.31	21.5	6.5	3.31
PSP 0.1	30	80	17	3.23	31.8	7.5	4.24
PSP 1.0	69	172	70	3.51	77.8	17.3	4.50
PSP 1.1	95	247	120	3.86	115.5	23.8	4.85
PSP 2.0	100	277	138	4.15	128.8	25.0	5.15
PSP 2.1	120	345	304	5.41	192.0	30.0	6.40
PSP 3.0	140	437	538	6.96	278.8	35.0	7.97
Average Ratio Secondary/Primary				4.20			

Table 2: Number of Fields for Primary and Secondary