*Malaysian Journal of Computer Science, Vol. 18 No. 2, December 2005, pp. 11-27*

# AUTOMATING A MODIFIED PERSONAL SOFTWARE PROCESS

***Mohd Hairul Nizam Md. Nasir and Azwina M. Yusof***
Department of Software Engineering
Faculty of Computer Science & Information Technology
University of Malaya
50603 Kuala Lumpur
Malaysia
email: hairul@fsktm.um.edu.my
azwina@um.edu.my

***ABSTRACT***

*Personal Software Process (PSP) is a defined software development 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. Even though several published studies have suggested that adopting PSP results in improved size and time estimation, and improved numbers of defects found in the compiled and test phases of software development, nevertheless not every software engineer adopts PSP in the process of software development .This paper attempts to clarify the issues that influence the adoption of PSP and explain how an automated tool can address these problems. The PSP adoption issue has been identified to be due to four reasons; 1) overhead in data collection and analysis, 2) excessive use of forms combined with a lack of a fully automated tool, 3) freezing of process definition, and 4) privacy issue. This paper describes an automated web based application tool to support the adoption of PSP in software development, called PSP.NET. This fully-automated tool can simplify both data collection and analysis to make this discipline more manageable and organized, with added features to make it more flexible, such as anti freezing of process definition, privacy support and collaborative sharing of defect information.*

***Keywords:*** ***Software Process, Personal Software Process, Automated tool***

## 1.0 INTRODUCTION

*"During the summer of 1996, TIS introduced the PSP to a small group of software engineers. Although the training was generally well received, use of PSP in TIS started to decline as soon as the classes were completed. Soon, none of the engineers who had been instructed in PSP techniques was using them on the job". Webb, et al. [1]*

In this day and age, computerized systems are being widely used throughout government agencies and information technology based companies to sustain and speed up their daily operations. While hardware components are an essential requirement of a computer system, without the software components, a computer cannot be used to solve specific business problems. Currently, there are many problems that occur during the development of a software project that lead to delay, over budget, customer dissatisfaction and poor quality of the software built. Inspired by the efforts of Deming [2] and Juran [3], the software engineering community has realized that a high-quality software development process will produce high quality products. It is generally agreed that "the quality of a product is largely governed by the quality of the process used to build it" Paulk, *et* al. [4]. Therefore, controlling and improving the processes used to develop software have been proposed as a primary remedy to these problems.

Process standards such as Software Process Improvement and Capability dEtermination (SPICE), ISO 9000 and the Capability Maturity Model (CMM) [5] have been proposed to assist organizations to achieve more predictable results by incorporating proven standards and procedures into their software process. Organizations that have made use of these standards advocated in ISO 9000 and CMM have usually shown excellent improvements. For example, by "improving its development process according to CMM 'maturity', Hughes Aircraft improved its productivity by 4 to 1 and saved millions of dollars", Pfleeger [6]. However, most of the process improvement initiatives consist of top down approaches that focus on an organizational level context. In contrast, Watts S. Humphrey [7] initiated Personal Software Process (PSP) that is more focused on the individual level context.

This paper will look into the issues that influence the adoption of the PSP in software development in both the education and industry sectors. It concludes with existing research that an automated tool is needed in order to make

the adoption of PSP more manageable and effortless. This paper also describes a proposed automated tool, PSP.NET that was built to solve these identified problems. It explains the five main features that are incorporated by PSP.NET in order to overcome the limitations of existing tools; i.e. simplified data collection and analysis, platform independence, anti freezing and flexibility, privacy support and collaborative sharing of defect information.

This paper is divided into 8 sections. The first section introduced the importance of having a standard software process and gave examples of existing software process standards that are used on an organizational 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 PSP in the industry and education sectors. The fourth section stresses on the importance of having an automated PSP tool and the five features that the PSP.NET incorporates. The fifth section describes the system architecture of PSP.NET while the sixth section provides an overview of possible future enhancement of PSP.NET. The seventh section gives an evaluation between PSP.NET against SEI guidelines, while the last section summarizes the main points of this paper.

## 2.0 OVERVIEW OF THE PSP

The PSP [7], pioneered by Watts S. Humphrey of the Software Engineering Institute (SEI), is a defined software development 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. PSP provides a defined personal process, guides software engineers in collecting and recording data, and defines ways to analyze data and make process improvements. The purpose of PSP is to help software engineers to learn and practice those software methods that are most effective for them. By defining, tracking and measuring work, software engineers will understand what they do and this will enable them to identify and choose the best methods and therefore see how they can consistently apply these methods when developing software. By practicing a customized set of orderly, consistently practiced, along with high quality personal software processes, software engineers will become more effective members of their development team and projects.

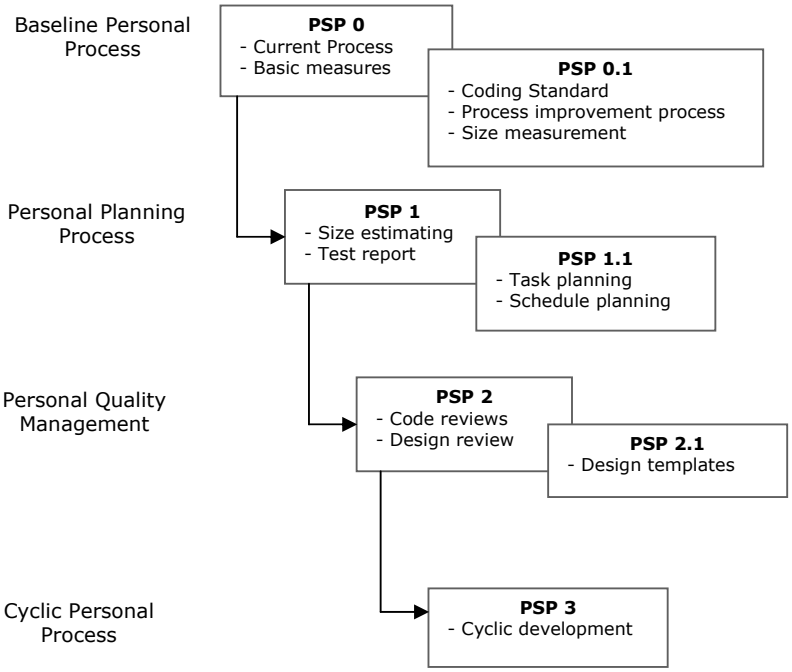



Fig. 1: The PSP Process Evolution

As shown in Fig. 1, the PSP follows an evolutionary improvement approach. A student or professional learning to fully integrate PSP into their process begins at Level 0.0 and progresses in their process maturity through seven levels up to Level 3.0 with each higher level building on the prior level. Each level incorporates new skills and

techniques into their process; skills and techniques that have proven to improve the quality of the software process and to improve the estimating accuracy of the software engineer.

While PSP is a relatively young discipline that was introduced in 1995, many researchers have performed studies on the use and payback of implementing PSP in education and industry. The results of these studies are varied. In 1997, Hayes and Over [8] performed an extensive empirical study with 298 engineers that focused on five dimensions of improvement: size estimation accuracy, effort estimation accuracy, product quality, process quality and personal productivity. The results of the study were impressive. In all cases, the data showed a positive change in the engineers' performance without any loss in productivity. The PSP can also aid engineers to achieve improvements in cost and schedule, product quality, cycle time and organizational process improvement. These four elements are very important from a business point of view, as supported by Pat Ferguson *et* al. [9]. The study was focused on the adoption of the PSP in three industrial software engineering groups from three different companies; Advanced Information Services, Motorola and Union Switch and Signal. The study shows that a good adoption of aspects of the PSP improved schedule estimation, and led to strong quality improvements in the developed software.

On the other hand, a study performed by Disney [10] reported that one of the general problems when implementing PSP is the quality of the data recorded in the collection and analysis stages. It was found that incorrect data recorded will lead to incorrect analysis of said data, and in the case of the PSP, can cause a ripple effect. Errors caused by incorrect data not only cause incorrect values in the current project but may ripple through to future projects, even if all subsequent calculations are done correctly. Another study performed by Barry Shostak *et* al. [11] reports poor adoption of the PSP in industry. They studied 28 software engineers who were taught PSP and also surveyed some engineers that had taken the PSP training from McGill University. The study found that only 46.5% of the engineers kept using the concepts from PSP.

## 3.0 PSP ADOPTION ISSUES

PSP as originally described by Humphrey [7] is a manual process. Disney in her thesis [10] termed manual PSP as the case where the PSP forms must be filled out by hand, either by editing a copy of the form on-line, or by filling out a printed copy with pen or pencil. Even if tools such as spreadsheets are used to collect historical data and to provide various computations, if they do not automatically insert and maintain the correct calculated values in the appropriate places in the forms, then the technique is still considered as manual. This section attempts to clarify the issues that influence the adoption of the PSP and explain why PSP is not adopted fully in both settings.

### 3.1 Overhead in Data Collection and Analysis

Since PSP is a manual process, it requires software engineers to create or print out forms in which they manually record all the significant measures taken regarding log efforts, defects that have been injected and detected during software development, the size of the software, and other related measures. As a lot of effort is put into producing a high quality software product, most software engineers find it troublesome to manually record data into the PSP forms, e.g. recording defect data into the printed Defect Recording Log form. In addition, manual PSP requires them to keep a stopwatch in order to record the interruption or defect removal time. Additional forms will use this information 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. Also, a large number of different documents and forms need to be managed and organized properly so that they are easily accessible whenever they are required.

Table 1 illustrates the approximate number of fields for each PSP level that needs to be filled by software engineers. As shown, when software engineers are in PSP Level 3.0, they are required to fill-in approximately 1115 fields. If you use the assumption that one field takes an average of 10 seconds to fill-in, it will take a total of 186 minutes for engineers to completely fill-in all fields. Also, the process of collecting data, regardless of effort and defective data, and the calculation of measures for the PSP 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 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 "1.00p.m to 4.00p.m, working on project 'A'" in the testing phase. In theory, it seems simple

and trouble-free, 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 a different task or project. Problems occur when changes need to be made to an entry or process. If the changes are small or minor, e.g. renaming the phases, it may not prove to be that difficult. But for bigger or major changes, i.e. changing the size measurement or estimation technique, it may prove to be very complex as it may affect many forms and scripts. This is because the PSP data is dependent on each other and correlations exist between the forms. Software engineers must locate the PSP forms for their project, and then reference various values, e.g. at least 45 times for PSP2.0, not taking into account size and time estimation.

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 analyses 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. Adding an additional measure to software engineers' work product and processes may have a huge psychological overhead to the engineers, and thus, the possibility of errors during data collection and analysis are high. This is proven by Disney's thesis [10].

Table 1: Number of fields in the PSP Process Level

| **Process Level** | **Approximate Fields** | | | | | | **Total** |
|---|---|---|---|---|---|---|---|
| | * **Header** | ** **Log Form** | *** **Summary** | **** **PIP** | **** **Templates** | ****** **Checklist** | |
| PSP0 | 14 | 12 | 60 | 0 | 0 | 0 | 86 |
| PSP0.1 | 23 | 12 | 79 | 13 | 0 | 0 | 127 |
| PSP1.0 | 52 | 12 | 89 | 13 | 145 | 0 | 311 |
| PSP1.1 | 76 | 12 | 100 | 13 | 145 | 116 | 462 |
| PSP2.0 | 88 | 12 | 170 | 13 | 145 | 147 | 515 |
| PSP2.1 | 140 | 24 | 197 | 13 | 247 | 147 | 768 |
| PSP3.0 | 198 | 50 | 460 | 13 | 247 | 147 | 1115 |
| **Total** | | | | | | | **3384** |

* 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

### 3.2 Lack of a fully-automated tool

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 needed 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. Based on the PSP training program feedback conducted in Ireland [12], it was suggested that an automated tool is needed to simplify the recording and analysis work of data needed in the PSP. 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, making data more convenient and easier to organize, and also to achieve a paperless working environment.

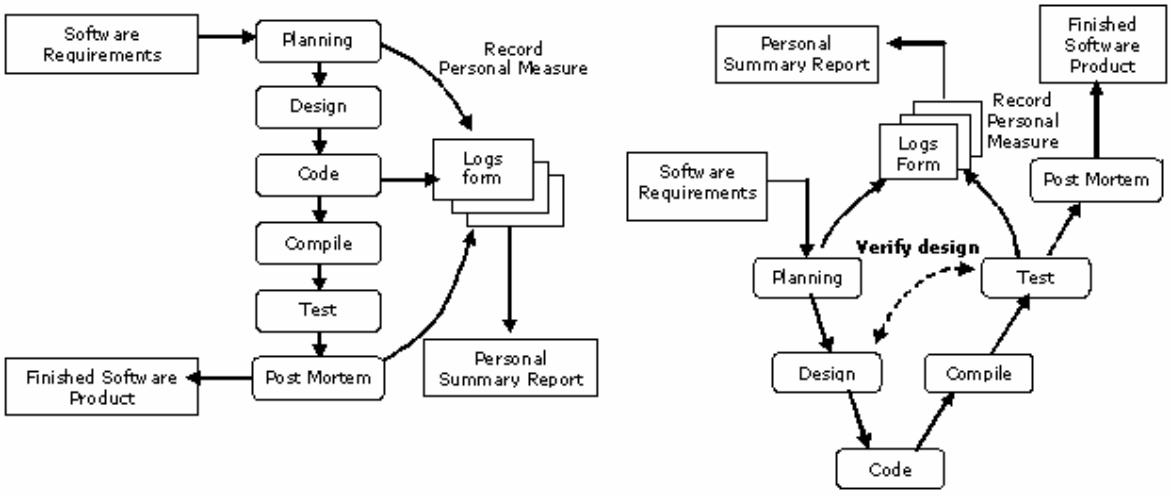
Soon after the PSP was introduced, a number of tools were developed to support data collection and to simplify the analysis stage. These tools are either fully automated or partially automated. Here, fully automated means the tool automates the entire process of the PSP including the collection and analysis stage, e.g. psptool 0.6 [13], PSP Studio [14] and PSP Tool [10]. On the other hand, partially automated means the tool does not enforce the entire PSP process but automates different aspects of the PSP, i.e. it focuses more on collecting raw data such as the collection of time or defect information. TimeLog [15], loccdelta and PC LOC Accounting Tools [16] are examples of PSP partially automated tools. However, most of these tools are platform dependent and can still be improved. Table 2 shows the features that current fully-automated tools have in comparison to each other.

Table 2: Comparison of Different Fully-automated PSP Tools

| | **Psptool 0.6** | **PSP Studio** | **PSP Tool** |
|---|---|---|---|
| **Support Level** | PSP 2.1 only | All PSP Level including PSP templates, reports and proposal | PSP0, PSP0.1 and PSP1 |
| **Portability** | X/Unix platform | Win32 platform | SCO Unix platform |
| **Flexibility** | Messages, text, Multi-languages configurable | Low in flexibility | Allow users to define their own new process, defect models and others, as well as customize the system |
| **Interface** | Simple GUI with only one mouse click for common actions | Simple and easy to learn | Character-based interface |
| **Storage Mechanism** | File format | Database format<br>SQL Anywhere | Database format<br>Progress 4GL/RDBMS |
| **Report** | No Report provided | Graphical analysis report for time, defect and effort, and PSP reports are outlined in *A Discipline for Software Engineering* book (Humphrey, 1995a). | Statistics on time and defects, and PSP reports are outlined in *A Discipline for Software Engineering* book (Humphrey, 1995a). |
| **User Privacy** | No | No | No |
| **Integrated LOC counter** | No (Input from user) | No (Input from user) | No (Input from user) |
| **Print function** | Yes | Yes | Yes |

### 3.3 Freezing of Process Definition

Originally, most of the PSP level practices fit the waterfall and V shaped software life cycle model as illustrated in Fig. 2(a) and Fig. 2(b) respectively, except for PSP 3.0 which deals with the spiral life cycle model [7]. Although this approach is 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. This is known as freezing the process definition. This limitation enforces process restrictions on software engineers, thus making the process improvement too impractical and inflexible. Moreover, in practice, most contemporary and large software development projects cannot be done using the waterfall or V shaped life cycle model that the PSP is based on, but they are more focused on risk oriented or object oriented life cycle model.




(a) (b)

Fig. 2: PSP Process Definition Model; a) The PSP Waterfall Model, and b) The PSP V-Shaped Model

In addition, PSP is more on code-centric development activities, whereby software engineers need to follow the process flow sequentially, starting from planning, designing, coding, compiling and finally testing the system, as illustrated in Fig. 2(a) and Fig. 2(b). But software engineers do more than just coding or programming. They are also required to perform non-programming activities such as system maintenance, writing reports, preparing technical

presentations and documentation, and etc. If a software engineer is asked to prepare technical documentation in relation to his/her programming work, how can he/she measure this since it is not covered in the PSP waterfall model? Another issue to consider is that the PSP splits the software development project into process levels only. In actuality, a software engineer may favor splitting his/her 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.

There are many diverse development process models used by software engineers such as rapid prototyping, spiral model, evolutionary development, incremental development 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. In other words, they prefer to use their own creativity to assist them in developing a better plan. Therefore, the improvement process model should not freeze but rather, it must 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 [7] intended that engineers should modify the PSP for their own needs, the PSP can be adjusted to fit any of the software developments models mentioned above.

### 3.4 Privacy

Originally, PSP is a manual process that focuses on individuals. It requires software engineers to collect their own data while providing useful and valuable insight into their personal development process. The engineers discover and learn their personal productivity, software development rate and quality, their average work per day and various statistics that the management could use to evaluate their work performance. Dellien in his master thesis [17] expressed his attempt to initiate a modified version of the PSP into an existing industrial setting. He advocated that using PSP in an industrial setting is different than implementing it in an academic setting for several reasons. One of the reasons is on privacy and integrity issues. If PSP data is made public, there is a temptation to view or alter that data, thus threatening personal privacy and integrity.

In a small software project, the anonymity and secrecy of data especially defects data for example, cannot be assured. This is because in a small software project, the development architecture and the allocation of the task modules are already known. Therefore, the programmer who wrote the code can be traced. The access to these very detailed and personal measures from the management level, such as managers, frightens software engineers. If management uses this data to evaluate their employees' performance, the employees may start to change their behavior to improve their measures. This may also unconsciously pressure their employee to produce the bests. For example, if the management orders the engineers to produce 100 lines of code (LOC) per hour but they only produce 60 LOC per hour, they might stop improving their code or optimizing the current system because they are more focused on doing as much coding as possible within a short period of time. In other words, they increase the quantity but at the same time decrease the quality of the software as a consequence from the measurement action taken by the organization. This kind of situation is termed as *measurement dysfunction*. As a result, the quality of the software product decreases over time due to the management action against this matter.

Thus, privacy concern upon these personal measures is very important. If ignored, it can lead to measurement dysfunction. If the improvement process model does not address measurement dysfunction, then the collected and analyzed data may not reflect what really happens, and changes to the development process may cause additional problem. If we employ PSP for motivational measurement rather than informational measurement, the objective of the PSP could not be achieved personally. Maybe, the organization may succeed in producing software products in term of quantity but not quality.

## 4.0 THE CALL FOR AN AUTOMATED TOOL

The drawbacks of manual PSP can be overcome through the development of a fully-automated tool. As Humphrey advocates, "Tools can be enormously helpful, but they alone will not fully solve software engineering's problems. Neither will process alone. Both are needed to obtain a balanced result. Improved tools, in fact, are needed in most aspects of the software process", [7]. Thus, a well designed fully automated tool is needed in order to simplify the entire task involved in the PSP. This section looks at how the proposed fully-automated tool, PSP.NET improves the weaknesses of the other automated tools by incorporating the properties mentioned below.

### 4.1 Simplify Data Collection and Analysis

Although the PSP requires software engineers to fill-in hundreds of data fields, in actuality, PSP data can be seen as consisting of two types of data: primary data and secondary data. Size, time and defects data are considered as primary data. This data type 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. PSP.NET is able to do as much collection of primary data as possible by providing electronic data forms which can be accessed on demand. Whenever possible, it also collects data automatically such as timestamp data. For example, users should not have to fill-in time log entry fields which can be captured automatically but instead, they should have the power to override or correct any of this automatically collected data. In this case, no stopwatch is needed and timing data can be made accurate to the second. All these personal measures are 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. PSP.NET automatically calculates secondary data and displays and inserts them into the electronic PSP forms. At this point, the calculation and analysis processes are done automatically; the time spent in both processes is minimized, thus reducing the analysis overhead. In addition, PSP.NET can also perform inter-project management by calculating and displaying all carry-forward values from a prior project to the current one.

### 4.2 Anti-Freezing and Flexibility

The problem of the freezing of process definition can be solved if software engineers have full control in determining their own development process. PSP.NET allows this by providing users with a user's customization feature that allows them to define their own development process and to follow it. PSP.NET does not enforce a specific development process model against software engineers like the original PSP does, but it offers them 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, PSP.NET supports this desire. In addition, the PSP.NET allows software engineers to incorporate other non-programming activities, e.g. 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.

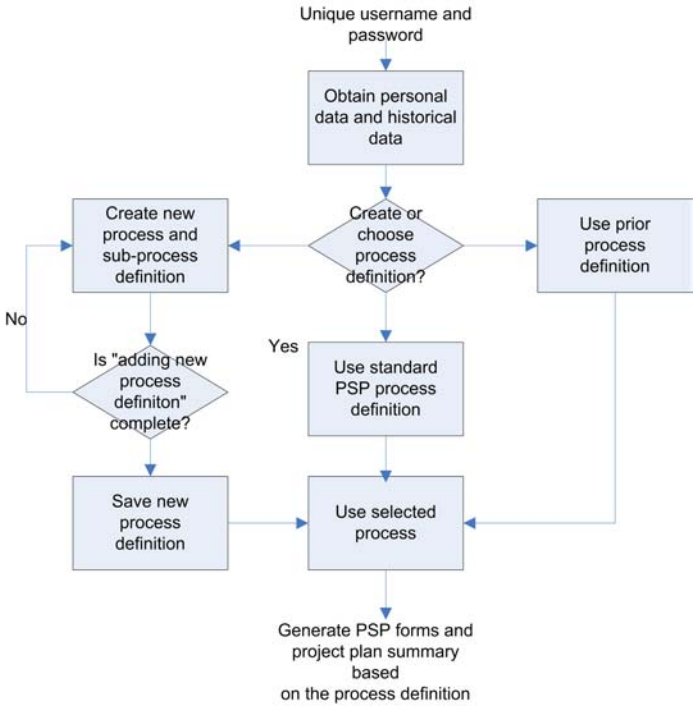



Fig. 3: Flow Chart for Process Definition Customization

Fig. 3 illustrates the process flow for the process definition customization of PSP.NET. For flexibility, users can either use standard PSP or use their own process definition. They can also create and customize new process definitions. PSP.NET also allows users to add new sub-processes in their process definition. The new process definition is then saved in the database once the users have completed the process. PSP.NET will then generate the PSP forms and documents, and continuously support data collection and analysis based on the users' choice of a process definition model.

### 4.3 Towards Platform Independence

Studies conducted by Barry Shostak and others on 28 engineers at CAE Electronic Ltd. ([11] and [18]) have pointed out the platform issue problem; there are various different platforms used by engineers in their software development work. For example, in one organization, some engineers might be using Visual C++ in developing their application which is based on the Windows platform, while others might use the GNU Compiler for C/C++ which is based on the Linux platform. If the automated PSP tool is built just for a Windows based-platform, how can the engineers working on a Linux platform make use of this tool?

Over the past few years, the World Wide Web (WWW) along with Hyper Text Transfer Protocol (HTTP) has been the most popular Internet access mechanism used by people around the world. Various web based applications have been developed to combat incompatibility legacy hardware and platform dependent problems. There are many compelling features supported by web technology, e.g. web browser software is all that is required to access web servers. Furthermore, web browsers are available for all major operating systems and the installation of web browsers at users' sites is very easy. Anyone with a UNIX workstation, or a Windows based or Macintosh platform; a modem, and protocol-compliant HTTP client software, such as Mozilla, Opera, Microsoft Internet Explorer, and Netscape Navigator, can make use of applications or services provided through the WWW. Therefore, the users' burden with incompatibility legacy hardware can be overcome, as reported in [19]. Since the WWW can solve the platform dependent and incompatibility legacy hardware issue, PSP.NET is developed as a web based application. Since web based applications are based on client-server architecture, the PSP.NET can be employed in three modes of operation: -

a) *Running through an internet connection*. In the first mode, there is a public server which stores the PSP.NET logic application, processes users' requests and communicates with the public relational database management system (RDBMS) to store and retrieve related data and users' personal measures. In this type of operation, the users make a request via HTTP through the electronic PSP form on their computer screen. The request will be processed by the web server and a corresponding service is then provided, e.g. providing electronic forms on-demand, data capturing and analysis, and synchronization. Here, an internet connection is a must. This mode is more convenient as the application tool can be accessed anywhere as long as there is an internet connection.

b) *Running on a local area network (LAN)*. The second mode can be employed if an organization does not have a public server or a public domain to store and maintain the automated tool, or if an organization desires this automated tool to run on their own LAN for security purposes. In this type of operation, no internet connection is needed but the corporate databases must be linked together with the web server in a manner that allows employees to access data through a web browser. Here, the degree of portability is lower as compared to an application that is running through an internet connection but nevertheless, it still remains platform independent.

c) *Running locally as a localhost*. The third mode is used when the engineers need to employ this framework for their self-improvement purposes. They may install the PSP.NET in their personal workstation, even if the company where they work does not employ this discipline. Some engineers do not want to store their personal data in a public office server for privacy reason. For that reason, the engineers are able to download the tool together with the web server and database management system and install them into their personal workstation. This type of operation makes their workstation as a web server and database server, and the tool is running locally in their computers.

### 4.4 Privacy Support

In using PSP as an improvement framework, personal measures are collected and this may lead to privacy and measurement dysfunction issues. The degree in which the privacy of individual data is protected is directly correlated to the possibility of measurement dysfunction occurring. If measurement dysfunctions occur, the data

collected and analyzed will not reflect the reality; whatever insights and analysis gained from this data will be faulty, erroneous, and troublesome. To avoid this problem, the individual data should be kept private and separated from organizational data by making the data anonymous or aggregating the data before it is shared.

To address data privacy and measurement dysfunction, PSP.NET has implemented the security measures below: -

- To secure the process data, a user can only access his or her own PSP data, and thus, to guard the access to the PSP database, the PSP.NET requires a valid personal password associated with a username for each user. The data can only be accessed by the user who owns the data. Furthermore, all the data gathered and analyzed during a process should be stored in a private database. Since each user is unique, the data of each user can be distinguished.

- Since the PSP.NET is a web-based application which is based on the client-server architecture, there is a public database located in the server. As a security measure, the users can choose whether to run HTTP with Secure Socket Layer (SSL) or run in normal HTTP mode. Moreover, the PSP.NET provides an alternative to allow users to download the server and run it locally. Therefore, all the data are kept under the users' immediate control and the process runs faster.

- All the access and request made by a client should always be verified and authenticate first to ensure that only authorized users can assess their own personal measures.

### 4.5 Support Collaborative Sharing of Defect Information

When developing software in a team, the defects that an individual software engineer discovers in his work product may be important and helpful to his other team members, just as the defects found by his team members can be of assistance to him in order to improve his own productivity. We can extract and manipulate all these defect data in order to investigate the most troublesome defects, defect type frequency, or where defects are injected and removed, and other relevant statistical information that can help us in our future software development projects. However, in the current PSP discipline, defect information is manipulated only on an individual level.

Consider a situation where a group of software engineers can share their defect information; they are able to share the defects that they have found especially the problematic ones, they can see what types of defects they are likely to find, and what is most important, they can share the defects that they failed to spot. This method is known as shared-defect information. Software engineers can combine the shared-defects information with the defects that they have found during their own software development project in order to gain helpful and valuable insight into their development process. This sharing of data will form one extensive library of shared-defect information which can assist software engineers in preventing the occurrence of the same defects that have been found by other team members. Furthermore, many software engineers tend to waste time in figuring out where and what a problem is and they often encounter difficulties when trying to fix the problem. But if an extensive public library exists, consisting of defect information along with their solutions, whenever software engineers find a defect, they can simply type in the defect type or error and the tool will search in the data repository to find similar defects. If a similar defect is found, the tool will then display the information and solutions regarding the defect. Thus, this will reduce the time it takes to fix a problem by learning from each other's experiences instead of providing many ways to solve one problem.

The shared defect information method can be efficiently implemented using an automated tool. As a core function, the PSP.NET supports group and multiple engineer environments. Users can share the defects they have found in one extensive public repository instead of a private library for personal use. Since the PSP.NET supports a collaborative environment, this mechanism can be implemented over an internet connection or a LAN environment, as illustrated in Fig. 4. When a software engineer finds a defect, he will record the defect data in the Defect Recording Log which consists of the defect type, the description regarding the error, and the proposed solution. These data are saved either in a private data or public data repository, depending on whether the users want to share the data or not. There is no way the data can be shared without the users' knowledge. The data in the public data repository can then be accessed by other team members in an anonymous and aggregate form to prevent measurement dysfunction. This public information will show and alert other users to take precautionary steps against the most troublesome or frequent defects in order to minimize the risk of making the same error by learning from others' experiences. In addition, when there is a problem fixing any defect, the users can search in the public database for a solution. We note that collaborative information and experiences are more helpful and robust than

individual data experiences because the more knowledge and information we share, the more we can learn in order to improve ourselves.

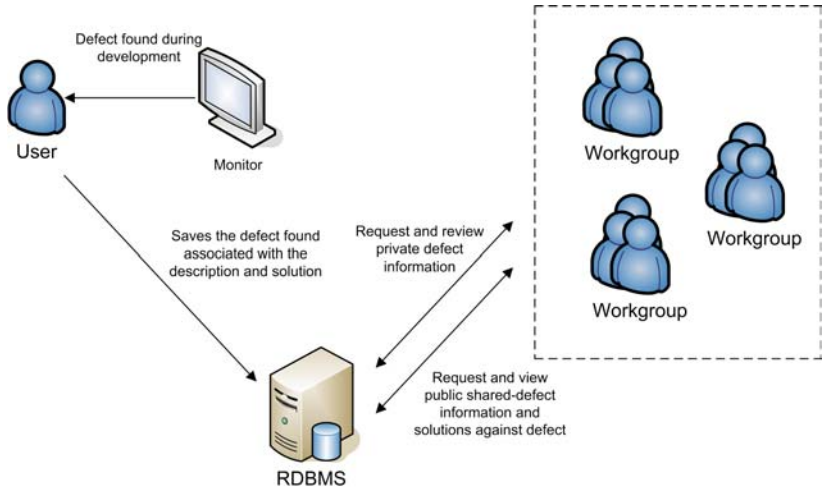



Fig. 4: Shared Defect Information through a Network

## 5.0 SYSTEM ARCHITECTURE

PSP.NET is a web based application which consists of two important layers; the application layer and the database layer, as illustrated in Fig. 5. Users can access PSP.NET using a web browser through HTTP or HTTPS protocol, whereby it can be accessed at http://www.pspdotnet.com while, the secure mode is provided at https://www.pspdotnet.com. PSP.NET is currently running on Red Hat Linux platform version 7.2, using Apache as a web server and MySQL as the database management system.

### 5.1 Application Layer

The application layer of the PSP.NET consists of the core programming logic which handles and manages all user requests. This layer is the core engine of PSP.NET which performs calculations, analysis and most importantly, communicates with the Relational Database Management System (RDBMS) for information retrieval and query. In addition, the application layer provides the front-end point for the interaction between the system and the user, displayed on the screen for the user to see. In this context, the application layer firstly generates the PSP forms, checklists and templates and then processes the users' requests by retrieving relevant data from the database, performs calculation and analysis against the data and finally sends feedback to the user in an appropriate order.

The PSP.NET application specific knowledge is implemented as data, server extensions, and scripts which are run by both the client and server parts. Since PSP.NET is running through a network connection, the application runs continuously and will not shut down until the user finishes his or her task. Or, at least some parts have to run continuously to serve other user. This mechanism is applicable for local area mode connection as well. The application layer is also accountable to ensure all the data stored are consistent and correct by validating them before they are stored in the database. Moreover, it also provides an exception handling mechanism. All the security measures and controls are also provided in this layer. A web server is given the responsibility to hold the application layer.

### 5.2 Database Layer

The database layer stores raw data which consist of users' information, password, online help and users' personal measures. In addition, the database layer is equipped with RDBMS to manage the data and allow communication between the database and application layers. The data are not stored in the client's computer. Although the client tier can implement parts of the functional core, the client itself can only store a limited amount of data related to the user interface. The database layer performs a query as requested by the application layer, retrieves the data from the database and sends it back to the application layer. The client and the database layer cannot communicate directly.

Any data transfer between them has to be mediated by the application layer. This provides for better application security and stability for the PSP.NET. The structure of the database must be designed carefully to avoid data redundancy as well as to ensure data consistency. A database server is accountable to hold the database layer.

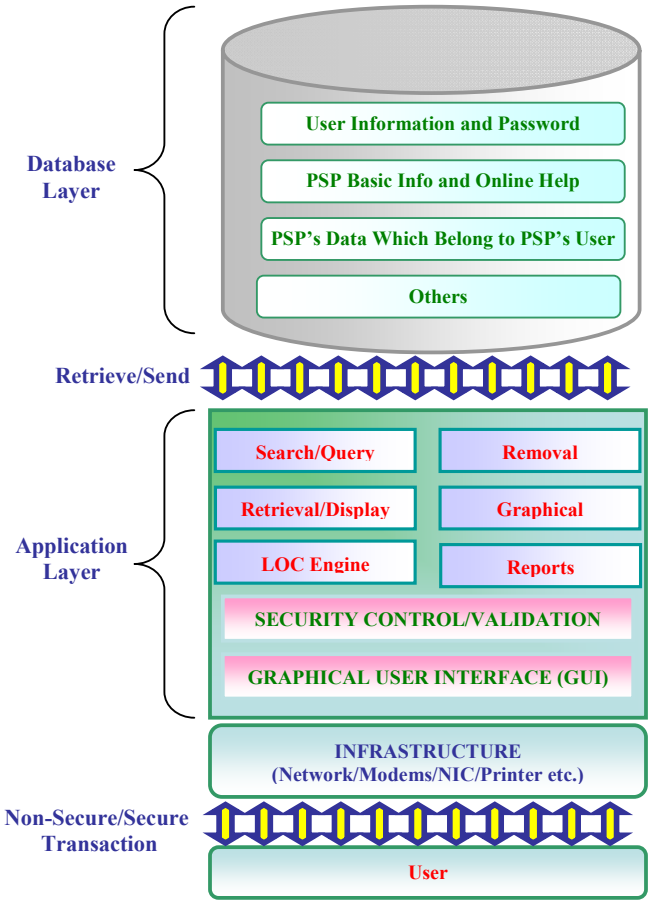



Fig. 5: PSP.NET System Architecture

## 6.0 EVALUATION WITH SEI GUIDELINES

The Software Engineering Institute (SEI) is in charge of the management direction as well as encouragement of the PSP initiatives. It has made available the principle guidelines for an automated support tool for PSP (SEI, 1996). This principle guidelines present suggestions and supply the system requirements to support the current PSP framework. Table 3 gives the comparative analysis of PSP.NET against the SEI principle guidelines. The analysis was conducted by taking key statements from the SEI guidelines and explaining how the PSP.NET tackles these statements.

From Table 3, it can be summarized that the PSP.NET supports most of the SEI requirements guidelines for a PSP automated tool. The SEI requirements guidelines that were fulfilled by PSP.NET covered all of the PSP level workflow starting from PSP level 0 until PSP level 3. PSP.NET reduces the overhead of software engineers' improvement by automating as much as possible data collection processes, and analysis calculations and conversions. Moreover, it also provides electronic data forms which can be accessed on demand for all PSP component modules. All of the PSP form can be published in HTML format and can also be printed according to the PSP levels.

Users are able to create their own process definition, and the development phases can be customized (whether it may be renamed or deleted) through an anti-freezing process definition mechanism. PSP.NET gives the users the flexibility to perform in any phase that they are comfortable with. To ensure privacy against personal measures,

PSP.NET is equipped with security features which have been discussed in detail in Section 4.4. PSP.NET has also fulfilled one of the SEI requirements by providing a built-in online help function for each of the PSP component including log form, templates, checklist, project summary as well as system help.

Table 3: Comparative Analysis of PSP.NET against the SEI Principle Guidelines

| Statement | SEI Recommendations | PSP.NET |
|---|---|---|
| Simplify Data Collection | The SEI recommends that PSP data is automatically copied between forms, eliminating the need to copy data twice. | The foundations of the PSP.NET is to minimize the overhead of software engineer improvement by automating as much as possible the data collection processes and analysis calculations and conversions Thus, the data in PSP.NET are automatically inserted and updated as required. Moreover, data consistency is also guaranteed, whereby the users do not have to copy the data by hand. |
| User Customization | The SEI recommends that the automated tool should provide user customization functionality. Specifically, it specifies examples of user customization such as the renaming of programming phases. | PSP.NET provides user customization through the anti-freezing process definition features. The user is able to create his or her own process definition, and the development phases can be customized, where by it may be renamed or deleted. |
| Standard Process versus Customized Process | The SEI recommends that the user get a full understanding of the concept and practicality of PSP and its framework before making changes to it to support a customized process definition. | PSP.NET gives the user control whether he or she needs to utilize the standard PSP process definition or utilizes his or her own process definition. PSP.NET does not enforce customized process definition. |
| Flexibility of Working on any Phase | The current PSP framework, tailors with waterfall software life cycle where it does not allow software engineers to move back and forth between phases. However, SEI recommends that the user be given the ability to operate in the phase of his choice. | The PSP.NET deals with this issue by providing a list of menu fields to display the list of development phases. The user can choose which phase he or she needs to work on at anytime from the list of menu fields. |
| Online Help | The SEI recommends an on-line help be made available with the PSP automated tool. | The PSP.NET provides an online help function for each of the PSP component including log form, templates, checklist, project summary as well as system help. |
| Data Privacy Statements | The SEI recommends that it is important that personal measures are private with the ability to provide public access to completed forms, while still maintaining that the engineer profile remain secure. | PSP.NET provides security measures as follows:<br>1. Provides an access authentication mechanism by assigning each user with a username associated with a password.<br>2. The users can run the PSP.NET in secure mode by providing SSL connection where the connection between their computers and the web server is encrypted.<br>3. All the access and request made by the client are always verified and authenticated first to ensure that only authorized users can access their own personal measures.<br>4. All the password stored in the database is in an encrypted |
| A Published Project Plan Summary | The SEI recommends that the PSP Project Plan Summary can be published based on the current PSP framework and the structure of the forms is different between each level. | PSP.NET provides a printing facility for each of the PSP form, checklists, templates and project plan summary that can be published in HTML format and can be printed. The structure is quite similar but the difference is only in the header information in each form where it displays the date of project creation and status of the project. |
| Estimation Task | The SEI supplies a statement regarding estimation of tasks. "To perform a programming task, the PSP requires that historical data are available (to predict productivity, defect injection/ removal rates and code size)". This statement promotes the PROBE method for the estimation of software size. PROBE takes two inputs: an estimate of the size of a new program and the user's historical database. | Since it is assumed that not every engineer may be comfortable with using PROBE. PROBE has been considered as an optional requirement method that is to be considered for future work. |

## 7.0 FUTURE WORKS

Based on the study and development experience encountered in this research, it is suggested that the PSP.NET can be scaled in terms of functionality. The following suggestions provide examples as to how the new system can be enhanced in this way.

- Automated data collection is not good enough to support the data collection. An enhanced and intelligent tool support is needed where it can automatically know what a user is doing and able to capture the time while the user is working. This feature can be provided by making the user's development environment responsive or use intelligent agents to collect the data automatically for the user. This feature can also identify what files are associated with which projects, have the ability to parse the source code and can detect some defects and know whether the user is busy or idle.

- Another feature that may be provided is implementing an intelligent agent. The intelligent agent can look and analyze the user's current and historical project data and report on interesting patterns that it notices. For example, the intelligent agent could report to the user when his or her productivity is outside of the average bound and assists the user by reflecting on what has changed in the current environment. Moreover, it could track the progress against user's estimation and alert the user if he or she is running over or under budget.

- A timer function can also be extended into the Defect Recording Log. In this context, the timer function is capable of capturing the time spent on fixing a defect, and then automatically inserts the value in the Defect Recording Log. This function will offer better and more accurate value for the new system.

- More advanced line of code (LOC) counter can also be used to improve the system in the future. The capability of the new automatic LOC counter is not limited to counting the lines of code only but also to keep track of the added, deleted and changed lines of code in multi-version programs. This can be very helpful in large software projects. The LOC counter will also automatically store all these values in the project Plan summary. For further extension, this LOC counter should also support multi-programming language type since there are many different programming languages employed by engineers in their software development work.

## 8.0 CONCLUSION

This paper highlights the research and development of a software tool support for a personal software development process namely PSP.NET. The basic notions and concepts were extracted from the SEI, which introduced the PSP concepts. 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. 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. The aim here is to formulate the PSP so that it is well-managed, effortless and widely adopted.

Thus, five main features have been proposed which have been incorporated in the PSP.NET. Firstly, as the core function, the PSP.NET reliably supports data collection, and automated computation and analysis of measures which will lead to an increase in the adoption of the PSP. Primary data are collected using electronic forms provided by PSP.NET or inserted automatically, while secondary data are automatically computed. Duplicate data need only be entered once, and will automatically be displayed on the PSP forms when needed. This will reduce overhead and the time spent during the recording and analysis stage.

Secondly, flexibility in defining the development process is also taken into consideration. PSP.NET gives the users flexibility in defining and following their own process definition. It does not enforce a specific development process model on the users. Nevertheless, if the users want to employ existing process models such as the PSP waterfall model, they may also do so. Regardless, the users should be able to define and customize their own processes, with the tool continuously supporting data collection and analysis based on the defined processes.

Thirdly, to avoid the problems of platform dependent and incompatibility legacy hardware issues, PSP.NET was developed as a web based application. Since web based applications are based on client-server architecture,

PSP.NET can be employed in three modes: running through an internet connection, running on a LAN, or running locally as a localhost.

Fourthly, PSP.NET provides security measures to assure users' privacy as well as minimize measurement dysfunction. The main security measure provided is an access authentication mechanism that requires a valid personal password that is associated with a username for each user to access his or her data and personal measures. PSP.NET also offers users an alternative whether to run on normal mode connection (http) or secure mode connection (https) environment.

And lastly, the automated PSP tool supports group and multiple-engineer environments whereby users have the option of sharing the defects that they have found in one extensive public library instead of a private library for personal use. The data in the public data repository can be accessed by other team members in order to learn from the experiences of others. Solutions to defects found may also be retrieved from this public library. This will decrease the time it takes for software engineers to solve any defect problems.

A list of the main modules or components of PSP.NET is given in Appendix A, Table 4, while samples of the user interfaces of PSP.NET are given in Fig. 6 and Fig. 7 of Appendix B.

In summary, the benefits of employing an 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] D. Webb, Humphrey, Watts, "Using the TSP on the Task View Project". *Crosstalk*, February 1999.

[2] W. E. Deming, *Out of the Crisis*, Cambridge, MA: MIT Press, 1986.

[3] J. M. Juran and F. M. Gryna, *Juran's Quality Control Handbook*, Fourth ed., New York: McGraw-Hill Book Company, 1988.

[4] Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process,* Addison Wesley, 1995.

[5] Mark C. Paulk, "The Evolution of SEI's Capability Maturity Model for Software". *Software Engineering Institute, CMM Evolution*, October 16, 1994, page 1.

[6] S. L. Pfleeger, *Software Engineering: Theory and Practice*, Upper Saddle River, NJ: Prentice Hall, 1998.

[7] Watts S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, MA, 1995

[8] W. Hayes and J. W. Over, "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers". *Software Engineering Institute, Pittsburgh, PA CMU/SEI-97-TR-001*, December 1997.

[9] Pat Ferguson, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya, "Introducing the Personal Software Process: Three industry cases", *IEEE Computer*, 30(5):24–31, May 1997.

[10] Anne M. Disney, *Data quality problems in the Personal Software Process*. M.S. thesis, University of Hawaii, August, 1998.

[11] Barry Shostak, "Adapting the Personal Software Process to industry". Soft*ware Process Newsletter #5*, Winter 1996.

[12] P.O'Beirne and J.Sanders, "Personal Software Process: Does the PSP Deliver its Promise". *Proceedings of Inspire '97*, Gothenburg, Sweden, 1997.

[13] PSPtool version 0.6. http://www.virtual.net.au/simtqc/description.html.

[14] Joel Henry, Personal Software Process studio: http://www-cs.etsu.edu/softeng/psp/, 1997.

[15] Timelog. http://www.kclee.com/clemens/java/timelog/

[16] PSP resources page at the University of Karlsruhe. http://wwwipd.ira.uka.de/_gramberg/PSP/

[17] Olof Dellien, *The Personal Software Process in Industry*. Master's thesis, Lund Institute of Technology (Sweden), Department of Communication Systems, November 1997

[18] Khaled El Emam, Barry Shostak, and Nazim Madhavji, "Implementing concepts from the Personal Software Process in an industrial setting". *Proceedings of the Fourth International Conference on the Software Process*, Brighton, England, December 1996.

[19] J. W . Erkes , K . B . Kenny , J . W . Lewis , B . D . Sarachan , M . W . Sobolewski , and R. N Sum. "Implementing, Shared Manufacturing Services on the World Wide Web", *Communications of the ACM. 39(2)*:34-45, 1996

**BIOGRAPHY**

Mohd Hairul Nizam Md. Nasir received his Bachelor of Computer Science (Hons.) specializing in Software Engineering and Masters of Computer Science (by full research) from University of Malaya in 2004 and 2005 respectively. He is currently serving as a lecturer is the Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya. His area of specialization includes programming languages, and software process. His research for his Master's dissertation concentrated on automating a modified Personal Software Process.

Azwina M. Yusof received her Bachelor of Computer science (Hons.) from University of Malaya and MSc. in Advanced Computing from Imperial College of Science, Technology and Medicine (LON) in 1999 and 2001 respectively. She currently serves as a lecturer in the Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya since 2001. She is interested in object-oriented technology, component-based technology, software reuse and distributed systems.

**APPENDIX A**

Table 4: A list of the main modules of PSP.NET

| Module | System/New Component | Component For PSP Level | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0 | 0.1 | 1 | 1.1 | 2 | 2.1 | 3.0 |
| Sign-Up | √ | | | | | | | |
| Access Authentication | √ | | | | | | | |
| Process Management | √ | | | | | | | |
| Project Management | √ | | | | | | | |
| Time Management | | √ | √ | √ | √ | √ | √ | √ |
| Defect Management | | √ | √ | √ | √ | √ | √ | √ |
| Project Plan And Cycle Summary | | √ | √ | √ | √ | √ | √ | √ |
| Personal Improvement Proposal | | | √ | √ | √ | √ | √ | √ |
| Test Report Template | | | | √ | √ | √ | √ | √ |
| Planning Management | | | | | √ | √ | √ | √ |
| Review Checklist | | | | | | √ | √ | √ |
| Operational Scenario | | | | | | √ | √ | √ |
| Specification Template Management | | | | | | | √ | √ |
| Issue Tracking Log | | | | | | | √ | √ |
| Line Of Code Counter | √ | | | | | | | |
| Graphical Report | √ | | | | | | | |
| Shared Defect Information | √ | | | | | | | |
| Online Help | √ | | | | | | | |
| Account Management | √ | | | | | | | |
| Download Management | √ | | | | | | | |

## APPENDIX B

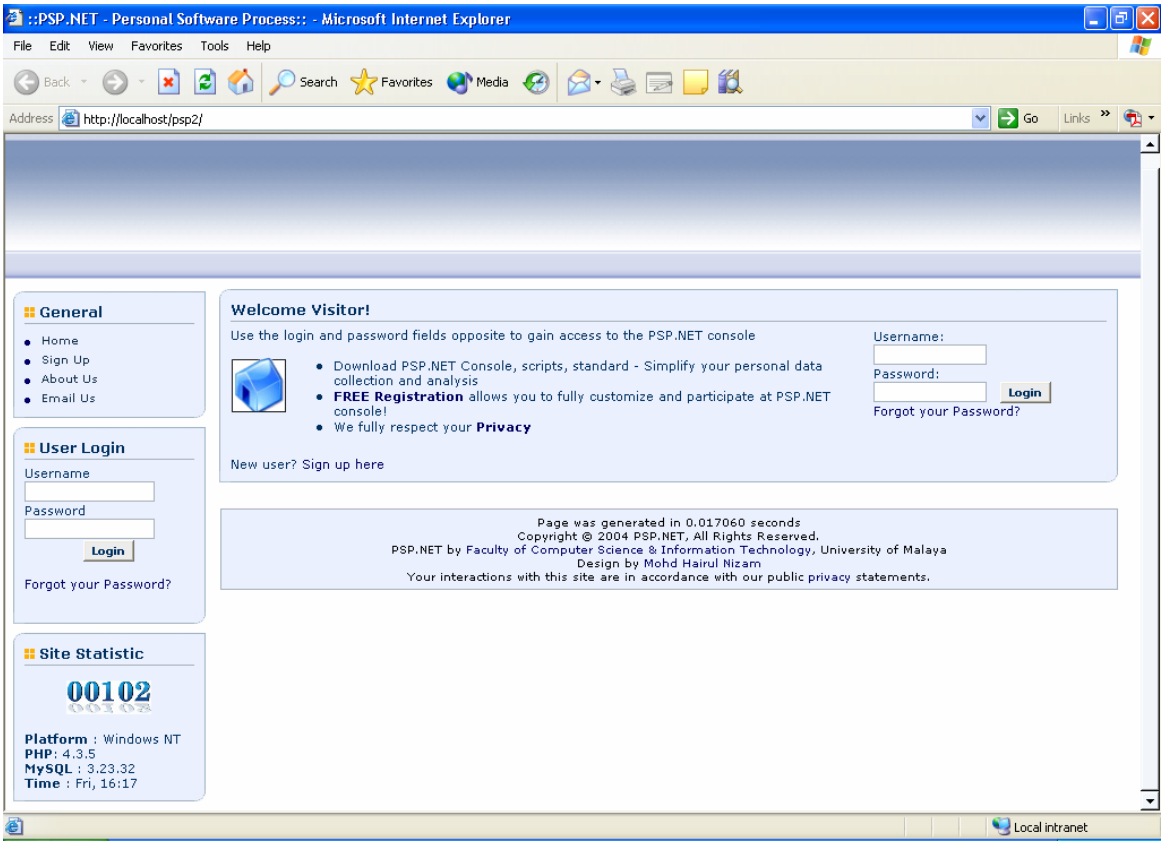



Fig. 6: PSP.NET Login Interface

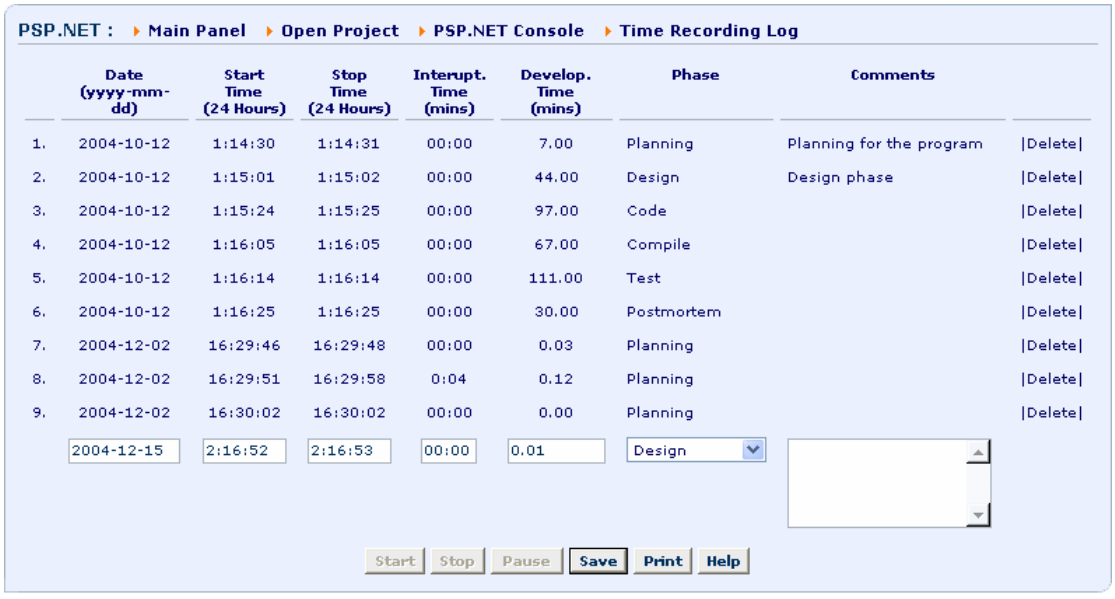



Fig. 7: Time Recording Log Interface in PSP.NET