International Journal of the Physical Sciences Vol. 6(21), pp. 4977-4989, 30 September, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.1114
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

Using agents to improve the usability of the PSP automated tool

Mohd Hairul Nizam Md Nasir[1]*, Salmiza Saul Hamid[2], Mustaffa Kamal Mohd Noor[1], Zarinah Mohd Kasirun[1] and Mohd Khalit Othman[1]

[1]Faculty of Computer Science and Information Technology, University of Malaya, Malaysia.
[2]Department of Advanced Research and Consultancy, Two Sigma Technologies, Malaysia.

Accepted 6 September, 2011

Various tools have been produced to assist personal software process (PSP) practitioners in implementing their processes, which require strong discipline from the individual software engineer. Nevertheless, most of the currently available tools still require software engineers to become involved in time-consuming manual processes and offer limited assistance. This research study presents the substantial potential for software agents to be incorporated into PSP automated tools by introducing four new agent-based features. These features are the proactive interface agent, an integrated Gantt chart with sensor-based scheduling, prediction ability and indirect management through multi-agent deployment. This agent has the additional features of flexibility and privacy. Integrated with the proactive assistant, the proposed tools are capable of collecting and processing accurate PSP data metrics and translating them into informative and meaningful information for both the software engineer and the project manager. The use of agents demonstrated in this paper is meant to significantly help engineers to practice all of the PSP processes effectively and in a timely manner and to get feedback on their performance with a visualisation platform at any time.

Key words: Personal software process, software agent, proactive assistance, software engineer, interface agent.

INTRODUCTION

In recent decades, the software development process has changed extensively. However, the gap between the demands of the software industry and the requirements of the development process in software delivery is still large. Despite considerable effort on the part of software engineer, software production is still sometimes delayed, costs can exceed the budget and user requirements are often not sufficiently met. For more than forty years, the software development experience has not succeeded in overcoming these problems. Researchers have persevered and been proactive at seeking a solution that improves the control of their software development and efficiently produces quality products.

For the sake of these goals, there has been increasing attention to the discipline of the software process to mitigate software development setbacks. Process standards, such as ISO 9000, capability maturity model integration (CMMI) and software process improvement and capability determination (SPICE) have been proposed and implemented to obtain more predictable improvement outcomes by incorporating these disciplines and procedures into organisations' software development processes. Because most process improvement initiatives are focused on the organisational level and are applied with a top-down approach, their strategies are less applicable when scaled down to the level of a small organisation. With the intention of providing a strong foundation for the adoption of the CMM framework, Humphrey (1995b) initiated the personal software process (PSP), which is focused on the point of view of the individual software engineer. The PSP provides a framework to assist software engineers to practice quality work, and it has been proven to dramatically improve the quality, predictability and cycle time for software-intensive systems. Various tools have been produced to assist PSP practitioners at implementing their processes, which require strong discipline and commitment from the individual software engineer. However, most of the

*Corresponding author. E-mail: hairulnizam@um.edu.my.

currently available tools still obligate software engineers to become involved in time-consuming manual processes and offer limited assistance, especially in properly visualising the actual performance and productivity of individual engineers. These limitations have placed PSP strategies at a huge disadvantage, in spite of their powerful potential and clear benefits.

This paper presents the idea of incorporating software agent features into a PSP automated tool. This research is aimed at increasing the usability of the PSP tool by incorporating four software agent features, namely; the proactive interface agent, an integrated Gantt chart with sensor-based scheduling, a prediction capability and a form of indirect management through multi-agent deployment; we also propose two additional features for the PSP automated tool, which are flexibility and privacy.

Overview of the PSP

PSP is a defined software development framework that is based on a process improvement principle with the goal of assisting individual software engineers in producing quality work (Hayes and Over, 1997). Through defined techniques and operations prepared in PSP scripts, PSP practitioners are guided thoroughly with the purpose that they learn the most effective software process methods, practice them and gradually improve their skills in managing their own work. Scripts in general provide a guideline to ensure that projects meet their requirements and drive PSP practitioners from the initial planning phase to the post-mortem phase of software projects. Together, various types of forms that represent each of the defined activities are provided to guide software engineers in their development works.

PSP emphasises a continuous measurement practice that requires individual software engineers to collect, record and analyse three elements in the software development activity, which are time, defect and size. These metrics offer functional value to software engineers and are used to statistically analyse their performance on a project. A PSP framework consists of seven levels; at each level, new skills and techniques are incorporated into the process (Humphrey, 1996). A cyclic development method encapsulates the whole PSP process to be certain that measurement and improvement skills can be built in a natural and consistent manner (incrementally ordered).

PSP facilitates the development of planning skills whereby engineers are trained to identify their tasks, to make estimates based on similar experiences and to judge how they are performing; all of these skills lead to accurate estimates. Humphrey (1995b) mentioned two types of planning; the first is based on a period of time and the second is based on the activity. Both types are important in the planning phase of scheduled work, recording the time and the defect measurements and generating an activity progress report. A summary of information is provided at the end of the post-mortem phase. With this report, engineers can revise the project to resolve defects, update their personal checklist and measure their performance by comparing their current and previous work. Plan and track work better, estimate time better, generate high commitment towards quality, ensure continuous process improvement, increase capability of individual and improve personal work process are supposed to be the results of PSP (Humphrey, 1995a). In other words, each new project can benefit from the data collected in past projects, as PSP provides insights into improving the planning, productivity and quality of future work.

There are seventy-six documents, which include forms, processes scripts and instructions that are packaged together in PSP to be used by software engineers to collect software performance data. To support and simplify the adoption of PSP practices, a number of automated PSP tools have been developed. These tools were initially built to support digital spreadsheets and computerised software process systems related to PSP data collection and analysis. The collection and analysis of all the PSP data are too difficult without the use of automated tools. The subsequent tool releases present upgraded versions that include higher analysis for digitally recorded information. A research study pointed out that integrated tools in a PSP are necessary to obtain high-quality PSP data (Disney, 1998). A single tool with all needed functionalities could help PSP users to focus only their work results rather than struggling with the complexity of the PSP process. In fact, the collection and analysis of all the PSP data are too tedious if performed manually (without automation). Although, the currently available PSP tools offer multiple choices of functions to help software engineers utilise the PSP and their usage is optional.

MATERIALS AND METHODS

A detailed study was conducted on the two main research areas, namely PSP and software agents. The notion of the PSP and its framework were studied and analysed, covering the analysis of PSP scripts, forms, procedures, formulas and tools, to obtain an understanding of the PSP operational framework. At the same time, this study was conducted in the area of software agents, which include concepts and taxonomies. The purpose of this study is to determine the possibility of incorporating software agent features into our proposed tool. We believe that by incorporating software agent features, an informative and proactive version of a PSP support tool can be built.

Subsequently, six widely discussed and/or recently released PSP tools have been selected and extensively reviewed. The tools are PSP studio (Design Studio Team, 2009), Dashboard (Team Process Dashboard, 2009), Hackystat (Johnson, 2001; Johnson et al., 2001, 2003), Personal Software Process Assistant (PSPA) (Sison et al., 2005), Jasmine (Shin et al., 2007) and PSP.NET (Nasir and Yusuf, 2005). The features and limitations of these tools were compared and analysed. This process led to producing twelve important features offered by tools, that is, environment, scripting, data collection, interface, planning wizard, lines of code (LOC)

counter, defect sharing, user available, user privacy, report, interface agent and metaphor. This comparative analysis is essential in identifying and deriving a set of features for our newly proposed tools.

Several solutions are proposed based on limitations identified in the currently available PSP support tools. These newly proposed solutions will become features of our new tool. PSP components and the new proposed features are examined and investigated separately, and they are treated as modules for the proposed tool. As a result, high level systems architecture for the proposed system was developed based on a 3-tier client-server architecture.

Finally, comparative analyses were made to compare our proposed tools to existing PSP tools based on the identified features.

Existing PSP tools

In this research, reviewing the six PSP tools enabled the subsequent comparison of features that are offered by each tool. The first PSP tool reviewed was PSP studio**.** This tool was developed in 1996 to 1997 as a project of the design studio team at East Tennessee State University (Design Studio Team, 2009). It automates all seven levels and forms of PSP and is very simple and straight forward in its implementation. PSP studio, however, is only supported by the Win32 platform and is driven by the SQL anywhere database system. Additionally, this tool lacks flexibility when the PSP process flow is programmed in 'fix' mode, which makes engineers unable to change the process according to their needs.

The second PSP tool, Dashboard, is a powerful tool that helps users to plan work and track progress easily. This tool was built by the U.S. Air Force in 1998 as a fully automated PSP tool (Team Process Dashboard, 2009). The main dashboard window is designed to be as small as possible to coexist with an integrated development environment (IDEs) and other developer tools. Built-in time and defect logs allow the user to view, filter and edit time and defect data. The process dashboard, on the other hand, supports planning and tracking, data collection, data analysis and data export. This tool is fully implemented in Java and runs on various platforms, such as Windows, Unix, Linux and Macintosh. Finally, the dashboard displays graphs and reports for performance analysis, which are available only at the end of the project completion phase.

Another tool, Hackystat, is an open source framework for software project data collection and analysis. Hackystat was developed in 2001 by a group from the University of Hawaii in collaboration with commercial software companies (Johnson, 2001). Using Hackystat, users typically attach software 'sensors' to the development tools that come with the framework. The sensors automatically collect metrics data from the engineers' activities and send the data to the Hackystat server to be analysed. Hackystat, however, is not compliant with every developmental environment tool, including notepad (Johnson et al., 2003). In addition, this appliance focuses merely on automated metrics collection and analysis rather than on the PSP processes as a whole (Shin et al., 2007).

The fourth PSP tool, personal software process assistant (PSPA), is a tool that was designed at De La Salle University to address the issue of recording difficulties. Its concepts were based on observed improvement feedback from a student with defect handling skills in an advance course (Sison et al., 2005). PSPA is a web system that is written in the .NET language and is combined with a plug-in agent written in JAVA. The ability to perform automatic recording of compile defects in a variety languages, such as C and JAVA, differentiates the tool from others.

The fifth PSP tool, Jasmine, was developed to help engineers perform the PSP (Shin et al., 2007) by automating data collection and providing enhanced support for planning and tracking. It has an electronic process guide (EPG), which allows easy navigation on PSP elements and an experience repository (ER), which allows process-related information to be stored and shared. Both EPG and ER integration help engineers to understand and use PSP effectively.

The last PSP tool reviewed here, is the PSP.NET, and was developed for a dissertation project at the University of Malaya (Nasir and Yusuf, 2005). This tool addresses all levels of PSP from PSP0 to PSP3.0, including all fields, log forms, checklists, templates and summary. The core purpose of PSP.NET is to simplify and facilitate the gathering of personal measures with an individual-centric focus and to provide direct access for analysis of the data collected (Nasir and Yusuf, 2005). However, the shortcoming of the overhead in the PSP data collection process still remains, because PSP.NET requires a significant amount of human intervention. Furthermore, the tool is inapplicable to the project manager.

Issues with the current PSP tools

Lack of visualisation

Existing automated PSP tools facilitate software engineers' tracking of their development work so that they can manage their schedules effectively. In the process, there are many computerised forms that must be filled out and organised, especially if the engineers are involved in many projects. This situation makes it difficult for the software engineers to plan and track their actual position on a project's individual task unless their work progress is automatically processed and updated every time they input new data into the forms. Although, recently developed PSP tools provide data analysis functions, meaning that the input (inserted) raw data will be automatically processed and generated as reports, these reports are available only at the end of each project phase completion. The reports include a project plan summary (PPS), time-related reports and graphical reports. If the engineers need to check on their performance or work progress in the middle of project development, a report/result must be manually generated. This specification demands extra time and commitment, not only from the individual software engineers, but also from the project managers, because they have to spend significant time visiting and speaking to their engineers whenever needed. Thus, currently available automated PSP tools are considered to be lacking in terms of the visual presentation of work progress and performance.

This problem can be addressed by transforming existing PSP tools into a visualisation plan to help engineers move quickly through the process of presenting their work progress. Additionally, it is practical if the project manager is given a space to monitor the software engineers' progress at any point in time without personally going to each developer's workstation or waiting until a progress report is sent to them. This ensures that they can make appropriate decisions or adjustments (if needed) as early as possible.

Conventional GUI

Current interaction with an automated PSP tool is via a conventional graphical user interface (GUI), which follows an interaction paradigm known as 'direct manipulation'. In this paradigm, a PSP program will only do something if it is explicitly directed to perform that particular task, for example, by clicking on an icon or selecting an item from a menu. We follow the instructions and click the button to fill in the form or perform work without having an assistant to give advice or recommendations on the required actions. Timely predictions are pointed out as a key to improve the effectiveness of the whole software process (Ardil and Sandhu,

2010). A PSP automated tool is typically organised according to generic process functions rather than the context of the task and situation (Hassan et al., 2009). It is clear that direct manipulation interfaces offer limited usage in terms of how PSP practitioners interact with the tools to overcome their overhead in the PSP practices. This restriction is applied to the PSP tools reviewed here.

Rigidity in phase flow

The final issue with existing PSP tools is their rigidity. The PSP can be divided into six phases: planning, design, coding, compiling, testing and post-mortem (Humphrey, 1995b). Although, these approaches are generic, they do not allow software engineers to move back and forth between phases. Instead, software engineers have to move sequentially through each phase that is appropriate for the process. For example, when a user is in the coding phase and then discovers a plan defect, he or she cannot go back to the planning phase to fix the defect (Hassan et al., 2009). Disney (1998) 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' (Disney and Johnson, 1998).

This limitation forces process restrictions on software engineers, which makes process improvement impractical and inflexible. In a development environment, such as IBM's visual age for Java, the code is automatically compiled as it is saved, which makes the compiling phase unnecessary.

Hence, the tools for an improvement process model should not congeal, rather should support the practice by offering flexibility in their process flow so that the process can be conveniently used by the user. Currently, only PSP.NET provides this flexibility.

PROPOSED AGENT-BASED TOOLS

Agent-based technology has recently received a high level of interest, and is becoming popular in the software engineering field, particularly due to its outstanding potential for the development of application tools. For example, Microsoft has employed agent technology in their office application known as 'clipper' or 'paperclip', which is an interface agent that assists users by making it easier, more pleasant and more human for users to interact with Microsoft applications (Johnson et al., 2000). Another example is research conducted by Amiri and Shirgahi (2011) which recommended intelligent agents for electronic market buyer and seller to increase the effectiveness of computer decision making systems.

Inspired by the Microsoft paperclip agent (Maes, 1994), PSP support tools should also be upgraded to have informative and assistive responses to its practitioners in a natural way, such as through an interface agent expression (Hassan et al., 2009); this strategy would also reduce the manual intervention of humans and increase automation in all of the PSP processes.

Interface agent is one of the agent categories. The other categories are information, intelligent collaborative, reactive, hybrid and mobile agent. An interface agent can be described as a personal assistant to a particular application. Such an agent usually works by observing, gathering feedback from the user, receiving explicit instructions and communicating to other agents to collect information. Interface agents have the ability to learn and can trigger actions automatically when required (Maes, 1994).

Based on the shortcomings identified in the data gathering and analysis phase, we next describe the proposed features for the automated PSP tools.

Proactive user interface assistants

Current PSP automation is comparable to traditional command-line or menu-driven interfaces, which are known as conversational user interfaces. The user enters an input; the system reacts by receiving the input, makes the computation, records the result and displays the output. This method requires the users to manually read and analyse the result produced by the tool.

The visualisation information method can be effectively implemented using an interface agent, which supports the software engineer in the PSP environment. The user can see and view all the processes and their performance through the interface agent, which retrieves the information by communicating with other agents and a knowledge repository. Moreover, the interface agent proactively alerts the software engineers if their progress is too slow or needs more attention. At the project manager level, the software agent proactively updates the project status performance and the team member status performance based on the PSP metrics on a real-time basis. In this case, tracking data are always available in real time, and the main emphasis changes from collecting tracking data to making decisions with the data and taking corrective actions based on the tracking data. Moreover, based on the analysis of the current and previous project data, the interface agent will also proactively report any interesting patterns to the project team. This approach makes the automated tool more intelligent, and proactive assistants can wrap the tool with agent capabilities rather than producing only raw graphical reports at the end of a phase or project; these strategies are currently being implemented in some PSP automated tools.

Integrated Gantt charts with sensor-based scheduling

In the software development process, projects often fail because engineers are unable to see progress toward their goal as the project develops. Thus, it is important to guide engineers to ensure that a project runs as planned. The solution is a Gantt chart. The Gantt chart is a graphical representation that shows the tasks and milestones for a project. A Gantt chart is a well-known tool and is easy to build; however, Gantt charts are difficult to follow, mainly because the chart is rarely automated and is typically not built into an automated tool,

including a PSP tool. Although, the PSP is well-known as a method that can help engineers to observe their personal performance progress, it is difficult for the developer to visualise where they are with respect to a milestone or a checklist.

Our proposed tool has a built-in Gantt chart function, where the deadlines for every task keyed in by the engineers can be visualised in the form of a Gantt chart. This capability provides informative and meaningful information. For example, once a cursor is moved to project timeline in the Gantt chart, the time budget information will automatically appear to let engineers know how many days they should spend on each task or phase.

Prediction ability

Prediction ability means that an agent can analyse historical data and, thus, learn from previous experiences and provide recommendations and advice. With current PSP tools, engineers can view their project status using product plans. However, engineers must trace their progress through a report form or with a generated graph. It is difficult for software engineers to obtain information about their progress when completing a task. Moreover, no current PSP automated tools provide predictions to improve individual performance.

For example, when discussing the remaining duration and effort estimation errors in a software development project, Morgenshtern et al. (2007) used 'estimation error' to denote the difference between the estimated time and the actual time. Two types of estimation error were identified: overestimation and underestimation. When the actual results exceed the planned values, this condition is called overestimation, whereas underestimation happens if the actual result falls below the planned values. Both terms are familiar estimation terms; however, without prediction ability, the current system cannot inform the engineer about their condition and cannot provide advice.

In a normal system, there is no need for prediction in the system. However, when a system is related to human decision-making, this function could be very helpful. An intelligent agent observes and acts depending on the situation and the environment. Such an agent could collect and read data to predict and give suggestions to the user on what to do base on the information that it generates. As the agent collects more data, the agent becomes more intelligent. In this way, the agent-oriented concept is regarded as encompassing the strengths or capabilities of other paradigms, such as object-oriented and service-oriented paradigms (Akbari, 2010). In fact, this adopted concept in our research can surpass these methodologies when first, an agent possesses the common characteristics of an object that comes together with mental states (Iglesias et al., 1999). Therefore, the agent in an agent-oriented method is an intelligent version of an object in an object-oriented approach. Second, an agent not only performs service whenever service is demanded, but also explicitly acts according to its environment, meaning that it is capable of providing information or services whenever necessary (Jennings and Wooldridge, 1996). An agent can learn and grow within its environment and can even change its goal over time (Wooldridge and Jennings, 1995). By being predictive, an agent can perform tasks with minimal or no interference towards its goal due to its ability to interact with the environment or another entity as an information source. As a result, such an agent can provide a variety of information.

Indirect management through multi-agent deployment

In the coming decades, we expect to see increasing efforts to develop software that can perform large tasks autonomously and that can hide as many details as possible from the user. Rather than invoking a sequence of commands that causes a program to perform small, well-defined and predictable operations, the user specifies the overall goals of a task and delegates the tasks to the computer to work out the details. In the specification process, the user must describe tasks rather than just select them from predefined alternatives.

All of the processes in each phase are performed automatically, and the multi-agent interacts with other agents to fulfil the goal of the task. Multi-agents can be designed to consider the context of the task as they present information and take action. A collaboration of multi-agents is capable of flexible autonomous action in some environments. The multi-agent can be flexible in a PSP environment when the agent is reactive (maintains an ongoing interaction with its environment), proactive (takes the initiative) and social (interacts with other agents). Because of a basic change in the orientation of the metaphor, the agent-based PSP tool becomes an assistant rather than just a tool (Maes, 1994).

As the PSP becomes more and more useful to engineers in small organisations and for self-improvement, a more user-friendly human computer interface is needed for an increased marketable and commercial value. If the user could interact with the computer more naturally, the work could be done in a less frustrating manner (Hassan et al., 2009). Figure 1 shows the way that a multi-agent will be operated in our proposed tool. This figure will follow the original PSP procedure, which consists of an analysis, design and implementation process. The multi-agent that is placed between a standard PSP and the PSP's personal summary report will do its part to track and predict based on the input from the Gantt chart and other interface agent tools. Once the analysis is completed, the performance will be visualised for the users and will help the users to achieve

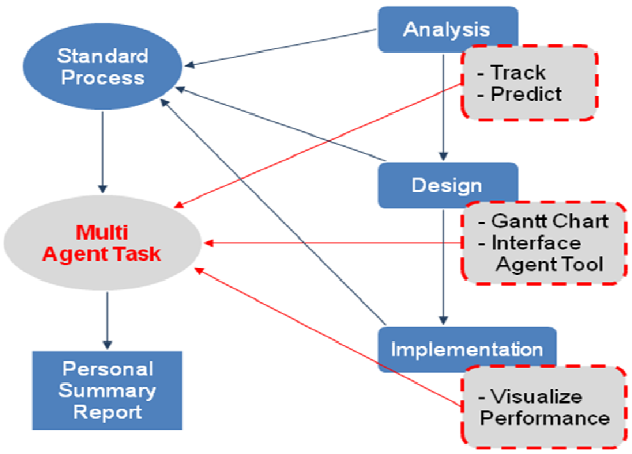

Figure 1. Multi-agent as indirect management flow.

their personal targets and to achieve better reports in the personal summary reports.

Additional features

Flexibility via phase customisation

The automated PSP tool should allow for phase customisation, which allows software engineers to define their own development process and to follow it. Moreover, this tool should not enforce a specific development process model on software engineers, as in the case with the PSP. An automated PSP tool should offer the software engineers full control and the ability to determine their own way of working. If software engineers do not desire to have a planning 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 customise their own defined processes, with the PSP tool continuously supporting the 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 processes as the processes in the PSP waterfall model, then they may still do so. The aim is to provide an alternative to achieve flexibility through user customisation. Our proposed tool offers flexibility by allowing the users to use either the standard PSP or to use their own process definitions. In other words, the users are able to create and customise 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. Furthermore, phase customisation allows the users to reorder and rename the phases. The tool will then generate the PSP forms, Gantt chart and documents, and continuously support data collection and analysis based on the users' process definition model.

Privacy

Evaluating software engineering performance is critical, especially with respect to project progress. It is the manager's responsibility to measure their performance. However, it is not always practical, given the time constraints, for a project manager to approach each of his or her co-workers to see their progress. At the same time, it is an ineffective way to identify the software engineers' performance exclusively after project completion according to the evaluation report. The assessment and monitoring of performance throughout a project is the key to producing a better product and can even improve the quality of the software engineering work. The proposed tool is intended to be a benefit to both engineers (software engineers) and managers (project managers).

A project manager is constantly in need of knowing the project status on a real-time basis, to report to their superior or clients. Unfortunately, it is not appropriate and might be interruptive to ask the engineers about their progress every hour. Moreover, the proposed system provides an efficient way to determine which developer works better than others with guidelines. In this situation, we see the usefulness of an agent; an assistant that can work at any time and tells project managers the current progress and performance of their engineers.

To maintain privacy and minimise measurement dysfunction (Nasir and Yusuf, 2005), two user types can be distinguished by user ID. Applying this control, the privacy of individual software engineers can be protected, because their private tasks are irretrievable by others. With the proposed tool, the project manager can monitor their software engineers without manipulating data or altering their work. Project managers may only view the progress of and provide advice or notes to the software engineers that they supervise.

High-level design of the proposed tool

Twenty-one modules will be constructed, as shown in Figure 2. The modules are derived based on three main components, that is, PSP components, agent features and tool administration.

The functional aspect of the proposed tool will include multi-agent features of three different agent roles: the task agent, search agent and interface agent. The task agent (TA) acts like a problem solver for the software engineer and will perform some performance calculations that are relevant to deciding what actions and states to consider while following goal formulations. A TA will

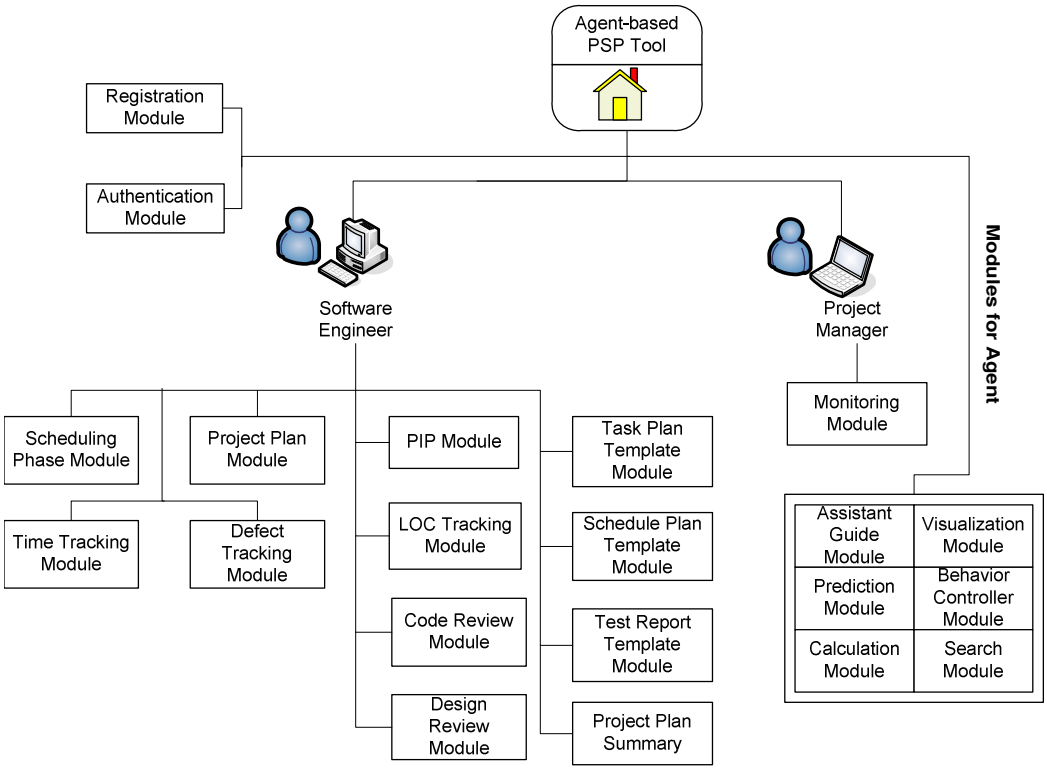

Figure 2. Modules of the proposed tool.

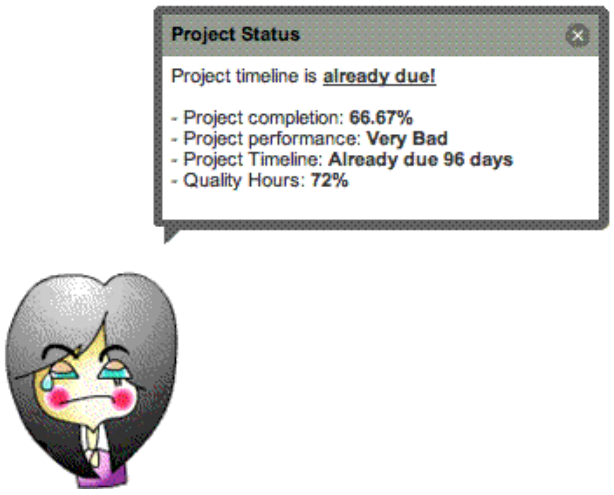

Figure 3. Sample of project performance output.

calculate and analyse project progress, will generate graphical results and will become an advisor that recommends a solution if an error occurs. A search agent (SA), on the other hand, provides intelligent access to a heterogeneous collection of information sources. Such an agent plays an important role in making sure that the agent system is successful in the searching process by probing into the database system. An interface agent (IA) will act as a presenter between an agent system and the end user that interacts with the user by receiving user specifications and delivering results. For example, once IA receives information generated by either the task agent or the search agent, it will display the result via a dialog box or an agent icon. Figure 3 provides a sample

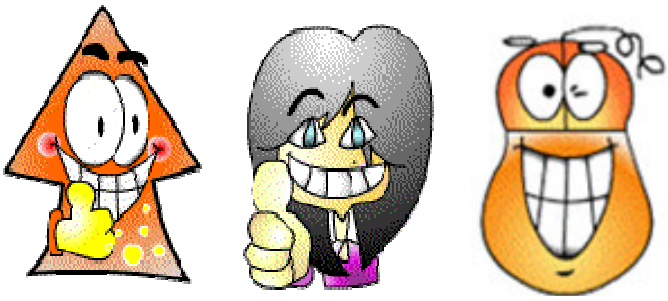

Figure 4. Samples of agent icons.

of a project status that is displayed using the IA. In general, the multi-agents in the proposed system are required to perform several tasks for the purpose of simplifying the plan and schedule, performing multi-tasking and visualising the defects and project progress as well as the performance of the software engineers.

An agent library will be created to handle the multiple agent actions and behaviours. In this case, a behaviour controller will be included that will play important roles in controlling the agent's appearance. This component will have a timer to set and will control the agent's animation, which is pleasing in appearance and sound. Once any actions are triggered to the agent, the IA will stop and reset the agent's animation and will assign relevant animations to the agent.

The design of the IA is to be specifically constructed to bring a positive tone or to lighten the mood of the software engineers. For example, in the case of good or excellent performance, the appearance of the IA that presents the message will be as shown in Figure 4. On the other hand, a poor performance outcome will use an agent icon that will advocate engineers to do better and to improve their performance. This simple approach, which touches on human psychology, is very well suited to the working environment of software engineers, who regularly deal with complex tasks. Based on the study of social models and interface agents literally and empirically, Rosenberg-Kima et al. (2007) indicated that the visual presence of an IA would definitely result in greater self-efficacy and positive attitudes toward engineering.

Figure 5 shows the agent's library of the proposed system, which is based on an If-Else rule when creating reactions to consequence results displayed by the system. Using the If-Else rule, four strategic steps will be developed into the agent's library.

Strategy I

"OBJECT" is a type of EVA, *"EVENTS"* is a module of EVA, *"ACTIONS"* is set of EVA's action, and *"EMOTIONS"* is all the emotion types in the agent's library.

Strategy I is the first step, where all the variables in the agent's library are declared. The type of EVA is called "OBJECT", the module of EVA is called "EVENT", a set of EVA's actions is known as "ACTION" and all the emotion types in the agent library are called "EMOTIONS".

Strategy II

Suppose that $x \in TARGET$ and t is time; then the state-space searching function Attr (x, t) returns a variable that represents which emotional state the target x is in at time t.

In strategy II, this step is the step that returns the variable that represents which emotional state the target x is in at time t.

Strategy III

Suppose that $x \in TARGET$ and t is time and R (x, t) represents the If-Else rules:

If the input is R (x, t) > 0
Else if the input is R (x, t) < 0
Else R (x, t) = 0; normal action.

Strategy III is the step that assigns an action for an agent through If-Else rules. This step is where the agent's policy and characteristics are set up to ensure that the agent acts as it should.

Strategy IV

IA_e finally returns to the emotional type of interface agent (*IA*):

$y = IA_e$ (Attr (x,t), R (x, t), ($y \in$ EMOTIONS)

Strategy IV is a final step in the agent's library process. In this step, IA will represent the result by the agent emoticon and complete the task of IA.

Cloud computing and internet applications have become popular; thus, we decided to develop the tool in a web-based environment. With the introduction of Web 2.0, it is no longer impossible to put a software agent into a web-based application. Most importantly, the problems of platform dependent and incompatibility legacy hard-ware issues can be accommodated (Nasir and Yusuf, 2005), which makes the proposed system applicable to multiple platforms. This tool will be useless as a user friendly tool unless it is easily accessible under various conditions and environments.

Hence, the proposed PSP tool will be implemented as a web-based application with three important layers: the presentation layer, the application layer and the database layer. The following items portray the structural and

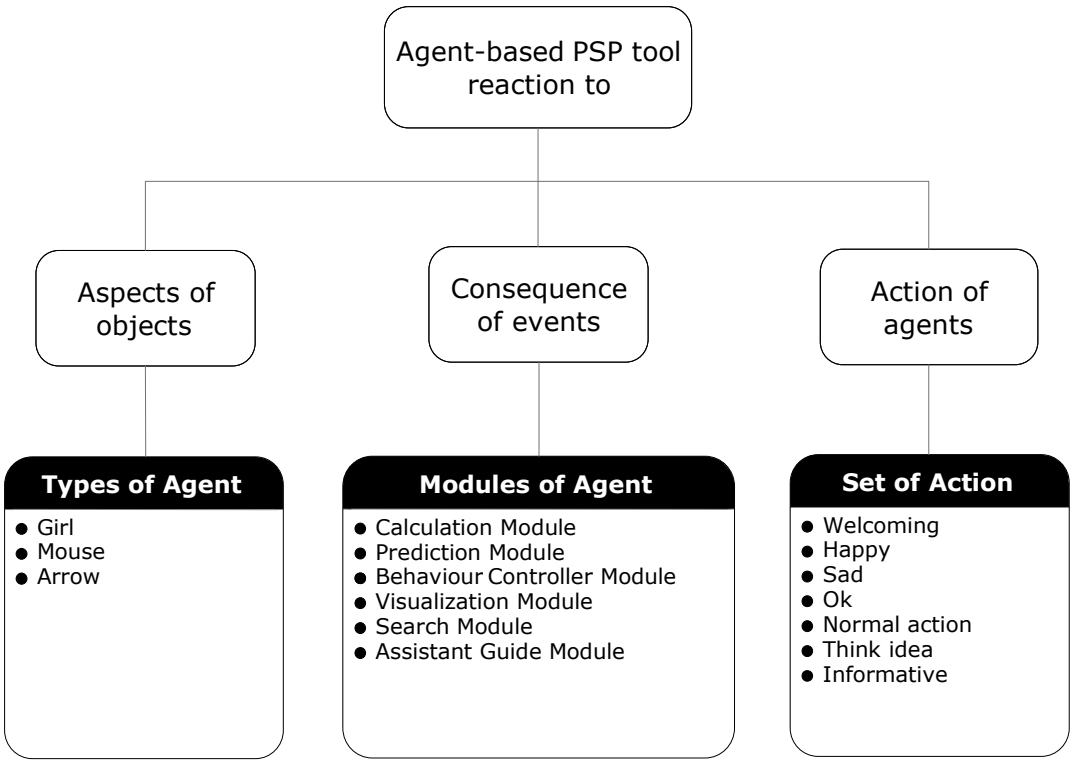

Figure 5. Agent's library based on If-Else rule.

functional aspect of each layer:

1. The presentation layer: this layer is the outermost layer in the proposed system, where the system user interacts with the system. The system user includes individual software engineers and project managers who participate in software projects. Different user roles will be granted to different sections of the system where the project manager is only allowed to view project progress and reports, and does not have permission to alter or manipulate the data.
2. The application layer: this layer is where the multi-agent system will be operated.
3. The database layer: this layer is the storage area for the proposed system, which will be composed of two storage sections; the PSP database data and the agent library.

The use of multi-agents is the core aspect in this research, whereby interface agent tools will provide conventional mechanisms for the agents to allow interactions or communications with the software engineer. Because the proposed system emphasises the visualisation aspect of the PSP automated tool, the name chosen is PSP-expert visualisation agent (PSP-EVA). By incorporating the multi software agent in this automation system, PSP-EVA not only is devised to be readily accessible to the application but also provides an interface informative agent for communication between the user and the application.

DISCUSSION

Previously, we highlighted six existing PSP tools. Here, we compare the existing PSP tools and the proposed agent-based PSP system, PSP-EVA. Table 1 summarises the similarities and differences among these tools under twelve criteria: environment, scripting, data collection, interface, planning wizard, LOC counter, defect sharing, user available, user privacy, report, interface agent and metaphor.

Comparative analysis of PSP tools

Web 2.0 enables software agents to operate in a web-based application system. This structure has the benefit that a software agent can act as an actor in a web system that allows network communication between its users. Actors represent anything that interacts with a system, and an actor can be an intermediary between a user and a system. According to Gillet et al. (2008), "an actor is any entity capable of initiating an event in the collaborative environment". The foundation of PSP-EVA is agent-based, which is a new paradigm in the area of

Table 1. Comparative analysis of different types of PSP automated tools.

	PSP studio	Dashboard	Hackystat	PSPA	Jasmine	PSP.NET	PSP-EVA
Environment	Windows only	Windows, Unix, Linux, Macintosh, etc	Eclipse, Visual Studio, Jbuilder	Open source IDE	Eclipse, Microsoft office, JBuilder	Windows and Linux platform	Windows and Linux platform
Scripting	Not stated	Java	Java	Java, C Language	Java, XML	PHP	PHP
Data collection	Manual and auto	Manual and auto	Auto	Auto	Auto	Manual and auto	Manual and auto
Interface	Simple and easy to learn	GUI	GUI	GUI and Pop-up windows	GUI	GUI & Pop-up windows	GUI and interface agent
Planning wizard	PSP template	Project plan template	Project plan template	Schedule planning template	Project plan template	Schedule planning template	Sensor based scheduling (Gantt chart)
LOC counter	No	Auto (but offline)	Auto (sensor-based)	Auto(Sensor-based)	Auto (Sensor-based)	Auto	Auto
Defect sharing	No	No	No	Yes (Auto)	No	Yes (Auto)	Yes (Auto)
User available	Developer only	Developer only	Developer only	Developer and project manager	Developer only	Developer only	Developer and project manager
User privacy	No	No	No	Yes (login and privacy task)	No	Yes (Login)	Yes (Login and Privacy Task)
Report	Graphical analysis report	PPS, graph and chart (estimation and defect)	Summaries and graph	PPS, graph (defect classification and ranking)	Test report summary	Graph, and Report summary	Agent visualisation report (summary and graph)
Interface agent	No	No	No	No	No	No	Yes
Metaphor	Not stated	Not stated	‘Toolbox’	‘Knowledge management portal’	Not Stated	‘Web-based’	‘Software agent-based’

software development process improvement. Compared to other PSP automated tools, PSP-EVA will be built in a web application platform using the PHP language. Thus, the system will be applicable to multiple environments and will span all connected devices. Regardless of its availability to both software engineer and project manager, the privacy of an individual software engineer is protected. Currently, only PSPA supports both the developer and the project manager, even though this feature is crucial, and only PSPA and PSP.NET have the privacy feature.

Some engineers of PSP automated tools include agent elements in their system that focus only on sensor-based inputs for tracking lines of code (LOC), but none of these engineers fully utilised this element for other purposes (Hassan et al., 2009). Indeed, the sensor-based inputs are only limited to the eclipse environment, as with Java, and are not compatible with the use of web platforms for PSP automated tools. PSP-EVA, on the other hand, provides sensor-based scheduling to improve planning quality. Planning is the first step in the PSP process after gathering and analysing all the requirements for the project. It is always difficult to track the plan. Although, PSP has a systematic process structure that helps engineers to plan and manage their workloads effectively, it is incomplete without flexible scheduling. Sensor-based scheduling is proposed to help engineers monitor the timeline of their project efficiently. It is believed that the overhead cost of utilising PSP in software development can be minimised if the scheduling is planned effectively.

The table shows that all the current PSP automated tools support report generation. However, the reports on a software engineer's performance and work progress are generated only at the end of project phase completion. The availability of the reports has been considered to be lacking due to improper data visualisation. Thus, software engineers still face difficulty when visualising their performance, which impedes their ability to maintain fast and high-quality work according to a project timeline. As an alternative, an autonomous, adaptive and semi-intelligent agent that provides support has the benefit of analysing current and previous project data and reporting interesting patterns. These features would make the user's development process responsive by assisting and reflecting on what has changed in the current environment. For example, an agent could report to the user when his or her productivity is outside of the average bound, when tracking progress against the user estimation and when alerting the user of whether he or she is running over or under budget. The agent is like a proactive mentor who can guide an engineer to improve his or her performance. Indeed, the application of a multi-agent system, where, for example, three agents communicate and collaborate in a coalition, would result in the fast generation of outcomes process (TSP) and project (Shamshirband et al., 2010). A coalition approach is indicated for improving task performance of autonomous agents operating in a common environment (Sarne and Kraus, 2005). Thus, PSP-EVA is projected to be able to predict, analyse and provide immediate suggestions or reports based on the current situation and environment. From there, decision-making and improvement processes can be performed faster.

In terms of functionality, Hackystat is an example of a PSP tool that can be easily expanded, because Hackystat can come out with another tool to broaden its functional coverage. For PSP-EVA, the application of an agent library paradigm during the design enables easy system expansion and contraction in the future. New agents, characters or transformation techniques to be added can be translated into different behaviours. New behaviours can be implemented by adding classes to the database of the agent library. Conversely, a behaviour that is no longer considered to be appropriate can be deleted from the system.

Finally, PSP-EVA will be the first PSP tool to implement an interface agent as a personal assistant to the software engineer. The central activity of a PSP in data collection for the purpose of tracking, measuring and analysing personal performance of an individual software engineer has required its practitioners to spend enormous amounts of time and effort to reach their intended benefits. This condition entails extra commitment, not only from the individual software engineers, but also from the project manager and management side, because enough resources and time must be allocated to measure and evaluate the empirical software project data. An interface agent will be an intermediate agent that allows a software engineer and project manager to communicate with other agents assisting them in performing their tasks or making inquiries during user access to the PSP-EVA.

The incorporation of a software agent in the PSP tool provides a significant advantage, by hiding the complexity of a PSP system using the agent. PSP methodology is a good example of how a complex system design can be simplified from a user's point of view by having an agent represent the system to the user. Therefore, the user would be more comfortable using a complex system without knowing the hidden processes with the assistance of the agent.

Generally, this solution reduces the overhead of adopting the PSP method, because the PSP implementation process is simplified and a substantial amount of time can be saved by using multi-agents to process, record and display data as compared to accomplishing these tasks manually. Subsequently, the accuracy of data processing and visualising ensures that there is a large quantity of data available. Therefore, software engineers can concentrate on their development projects rather than allocating a significant amount of time to collecting, analysing and computing data concerning their performance indicators.

CONCLUSIONS AND FUTURE WORK

This paper proposed an agent-based automated PSP

tool where software agent attributes are incorporated into the automated PSP tool, known as PSP-EVA. The purpose of PSP-EVA is to overcome the three core issues identified in the current PSP tools highlighted in this paper: lack of visualisation, conventional GUI and rigidity of phase flow, and the emphasis is placed on the effective visualisation aspect.

Six existing automated PSP tools have been studied and compared, and the comparative analysis shows that none of the tools use an IA as one of their features. An IA is capable of providing meaningful responses to a system's users and of learning the characteristics of its environment and triggering automatically when required. Such a capability would greatly help software engineers to comfortably use PSP tools as intelligent personal assistants. Moreover, the existing PSP automated tools still require much management commitment and effort. Looking at all the issues, the significance of incorporating an agent-based paradigm is emphasised by introducing a new agent (that is, a proactive interface agent), an integrated Gantt chart with sensor-based scheduling, a prediction ability and indirect management through multi-agent deployment with two additional functions: flexibility and privacy.

Using an agent concept, which reflects a paradigm shift in the software engineering world, we target the achievement of better information delivery from the computer to the user. Furthermore, a software agent has the potential for producing a paradigm that is closer to the real world and that provides a more accurate prediction capability as compared to traditional and object-oriented approaches. This approach also provides the benefits of visualisation and anti-freezing, which allow for an easy evolution of the system. Furthermore, the major functionalities in the PSP framework not only must be operated in an automation mode but they also must be attractive, supportive and persuasive (the presentation of the analysis result must affirm either the engineers' strengths or weaknesses) for the software engineers not to perceive the tool and subsequently the PSP processes as burdensome. Therefore, the inclusion of compelling elements in the PSP tool would be highly significant.

The ultimate aim of this research is to develop an agent in the automated PSP domain that incorporates the proposed features. Thus, future work would be to produce the proposed PSP-EVA, a new system that incorporates an interface agent, supports both the developer and manager, has privacy controls to prevent unauthorised users from accessing the data and, of course, has a friendly GUI, has a good usability and supports multiple platforms.

For future work, we are now upgrading the PSP automated tool to address software development at team level. In this context, we plan to deploy team software management body of knowledge (PMBOK) on top of PSP discipline as what has been suggested by (Nasir and Sahibuddin, 2011a), so that it can better address the software critical success factors (Nasir and Sahibuddin, 2011b). It is our hope that our proposed research here will complement existing studies in the area of software engineering, particularly in software process and software process improvement.

ACKNOWLEDGEMENTS

We are sincerely grateful to the Faculty of Computer Science and Information Technology, the University of Malaya and the Two Sigma Technologies Malaysia, for supporting this research effort.

REFERENCES

Akbari OZ (2010). A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance. J. Comput. Eng. Res., 1(2): 14-28.

Amiri S, Shirgahi H (2011). Designing buyer and seller intelligent agents in an electronic market based on emergency decision making. Intl. J. Phys. Sci., 6(6): 1244-1248.

Ardil E, Sandhu PS (2010). A soft computing approach for modeling of severity of faults in software systems. Int. J. Phys. Sci., 5(2): 074-085.

Design Studio Team (2009). PSP Studio. Available at: http://www.cs.etsu.edu/psp/

Disney A, Johnson P (1998). Investigating Data Quality Problems in the PSP. Proceedings of the ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering, Florida, USA.

Disney AM (1998). Data Quality Problems in the Personal Software Process. Master's Thesis, University of Hawaii.

Gillet D, Helou SE, Yu CM, Salzmann C (2008). Turning Web 2.0 Social Software into Versatile Collaborative Learning Solutions. First International Conference on Advances in Computer-Human Interaction. IEEE Comput. Soc., pp. 170-176.

Hassan H, Nasir MHNM, Fauzi SSM (2009). Incorporating Software Agents in Automated Personal Software Process (PSP) Tools. Proceedings of the 9th international conference on Communications and information technologies, Incheon, Korea, pp. 976-981.

Hayes W, Over JW (1997). The Personal Software Process: An Empirical Study on the Impact of PSP on Individual Engineers. Technical Report, Software Engineering Institute.

Humphrey WS (1995a). The Personal Process in Software Engineering. Proceedings of the Third International Conference on the Software Process, Reston, Virginia, pp. 69-77.

Humphrey WS (1995b). A Discipline for Software Engineering, Addison-Wesley.

Humphrey WS (1996). Using a Defined and Measured Personal Software Process. IEEE Softw., 13(5): 77-88.

Iglesias CA, Garijo M, Gonzalez JC (1999). A Survey of Agent-Oriented Methodologies. Proceeding ATAL '98 Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, London, UK. Springer-Verlag, pp. 317-330.

Jennings NR, Wooldridge M (1996). Software Agents. IEEE Rev., pp. 17-20.

Johnson P (2001). Project Hackystat: Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis. Available at: http://csdl.ics.hawaii.edu/techreports/01-13/01-13.pdf

Johnson PM, Kou H, Agustin J, Chan C, Moore C, Miglani J, Zhen S, Doane WE (2003). Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined. Proceedings of the 25th International Conference on Software Engineering, Washington, DC. IEEE Comput. Soc., pp. 641-646.

Johnson PM, Moore CA, Miglani J (2001). Hackystat design notes. Technical Report, Department of Information and Computer Sciences, University of Hawaii.
Johnson WL, Rickel JW, Lester JC (2000). Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. Int. J. Artificial Intell. Educ., 11(2000): 47-7.
Maes P (1994). Agents That Reduce Work and Information Overload. Commun. ACM, 37(7): 94-103.
Morgenshtern O, Raz T, Dvir D (2007). Factors affecting duration and effort estimation errors in software development projects. J. Inf. Softw. Technol., 49(8): 827-837.
Nasir MHNM, Sahibuddin S (2011a). Addressing a Critical Success Factor for Software Projects: A Multi-Round Delphi Study of TSP. Intl. J. Phys. Sci., 6(5): 1213-1232.
Nasir MHNM, Sahibuddin S (2011b). Critical Success Factors for Software Projects: A Comparative Study. Sci. Res. Essays, 6(10): 2174-2186.
Nasir MHNM, Yusuf A (2005). Automating A Modified Personal Software Process. Malaysian J. Comput. Sci., 18(2): 11-27.
Sarne D, Kraus S (2005). Cooperative Exploration in the Electronic Marketplace. Proceedings of the American Association for Artifical Intelligence (IAAA).
Shamshirband S, Kalantari S, Daliri ZS, Ng LS (2010). Expert security system in wireless sensor networks based on fuzzy discussion multi-agent systems. Sci. Res. Essays, 5(24): 3840-3849.
Shin H, Choi HJ, Baik J (2007). JASMINE: A PSP Supporting Tool. Proceeding ICSP'07 Proceedings of the 2007 international conference on Software process, Berlin, Heidelberg. Springer-Verlag, pp. 73-83.
Sison R, Diaz D, Lam E, Navarro D, Navarro J (2005). Personal Software Process (PSP) Assistant. Proceedings of the 12th Asia-Pacific Software Engineering Conference, pp. 687-696.
Team Process Dashboard (2009). Process Dashboard. Available at: http://processdash.sourceforge.net/
Rosenberg-Kima RB, Baylor AL, Plant EA, Doerr C (2007). The Importance of Interface Agent Visual Presence: Voice Alone Is Less Effective in Impacting Young Women's Attitudes Toward Engineering. Proceedings of Persuasive 2007, Stanford, California. Springer, pp. 214-222.
Wooldridge M, Jennings NR (1995). Intelligent Agent: Theory and Practice. Knowledge Eng. Rev., 10(2): 115-152.