Performance Monitoring Framework for Personal Software Processes

Muhammad Naeem Ahmed Khan[1], Mirza Aamir Mehmood[2], Mohammad Imran[3], and Raja Asif Wagan[3]

[1]Independent Researcher, Pakistan
[2]Department of Computer Science, Faculty of ICT, Balochistan University of Information Technology, Engineering & Management Sciences, Quetta, Pakistan
[3]Department of Information Technology, Faculty of ICT, Balochistan University of Information Technology, Engineering & Management Sciences, Quetta, Pakistan

ABSTRACT To reduce defect density in Personal Software Process (PSP), the current study introduced a novel PSP. Experiments were conducted using sample software codes to determine how to improve developers' productivity. Tasks performed by the developers were recorded into a log file on a daily basis to evaluate their performance. The experimental results helped formulate pertinent guidelines which can be used to improve software quality, performance, productivity, and efficiency of the developers. The study also defined distinct process levels, each equipped with comprehensive scripts, checklists, and templates to help developers adopt the mandatory steps in the PSP environment. The proposed framework may help automatically schedule and list activities of the individual developers. This study proposed a framework designed for consistent performance monitoring of software engineers within the PSP environment. For this purpose, a prototype time-logging model has also been developed under the Java platform that captures keystrokes and mouse clicks to automatically analyze the productivity of software engineers with respect to the tasks assigned to them.

INDEX TERMS performance monitoring, quality management, time-logging

I. INTRODUCTION

The Personal Software Process (PSP) serves as a structured methodology aimed at enhancing the performance of software engineers. Being a software development process, PSP helps software engineers improve their performance by using an orderly and data-driven technique. In other words, PSP is basically a method for software engineers to discover and develop their own development process [1]. PSP depends heavily on gathering and investigating personal data as a part of operative process execution. It helps professionals assess their performance when they are working in a self-directed environment. Furthermore, it provides disciplined personal structure to software engineers by helping them to plan, manage, and quantify their work. The key reason to using PSP is to build products with zero defects without time and cost overrun by helping software engineers to evaluate their performance and improve it using historical data. Therefore, historical data serves as a core facet to analyze, measure, and improve process performance. The four main elements supported by PSP data collection include measures, scripts, standards, and forms.

PSP, as an individual software development methodology, requires regular performance monitoring of software engineers to ensure timely completion of the projects. PSP has specifically been designed to assist

software engineers to improve their personal software development processes so that they may produce high-quality products. By following the specific PSP guidelines, software engineers can determine process deficiencies by performing meticulous analysis and then may produce a reliable estimate of product quality. This necessitates the support of an automated tool to collect reliable data as well as to address deficiencies of manual data recording, such as context-switching problems. In this regard, time-logging is an important aspect to measure productivity of software engineers engaged in designing, developing, reviewing, and testing software.

PSP comprises various methodologies which help software engineers in several steps of software development, such as development of software plans, quality measurement, and for effective organization of software products. Furthermore, PSP assists software engineers in effectively executing tasks encompassing software planning, software architecture, development of software, software test, and evaluation. PSP technology is tailored for individual developer work and is particularly suitable for workplaces that emphasize individual software development culture. The PSP methodology has been developed by Software Engineering Institute (SEI). It brings regulatory measures into the sphere of individual software development practices, aiming to enhance the value and prediction of systems heavily reliant on software. It involves fresh perspectives to enhance productivity and work quality, as the primary objective of PSP is to deliver software products free of defects, completed on time, and within the allocated budget. As a self-improvement framework, PSP encompasses a defined set of operations, along with assessment and analysis techniques. The PSP methodology is structured to be readily applicable across various stages of the process of software development. Likewise, this method can be utilized in context of several programming languages and software design practices including elicitation of software requirements, definition of processes, execution of evaluations, and defects removal. However, usage of PSP [1] should be complemented with Team Software Process (TSP) [2] to realize the overall aims and objectives of the software development activities. The quality of software products is closely linked to the quality of processes utilized during the software development phase [3], [4]. PSP offers a structured framework for the development of software, assisting the programmers in gauging and enhancing their personal productivity [4]. The PSP framework involves multiple levels, with each stage building upon the previous one through the addition of supplementary processes.

The current study proposed a framework designed to analyze the performance of software engineers in PSP environment. This study aimed to highlight the importance of time-logging-based performance metrics throughout the PSP software development process. Moreover, it also proposed a framework adapted to effectively monitor the performance of a development team. The study proposed a framework to monitor the performance of PSP team during the planning, designing, development, reviewing, and testing phases. The proposed framework is based on capturing keystrokes and mouse clicks during a specific time span for all the members of the software development team. The keystroke information is maintained in the form of structured logs that are analyzed automatically to

determine the amount of work done by different team members. The framework also generates alerts if the performance of any of the team members drops below the predefined minimum threshold.

The current study was conducted as a motivation to help facilitate software engineers to enhance their time management at work as well as to assist project managers to effectively monitor the performance of their team members, particularly in the PSP environment. The existing models pertaining to PSP performance monitoring software development activities in general, and the monitoring of individual performance in particular have been meticulously analyzed to propose a simple and easy-to-implement framework. The proposed framework is flexible enough to be scaled up to accommodate other performance-related considerations. The detailed description of the proposed model is explained in the next section. The current study aimed to measure performance, efficiency, and productivity of professionals as well as improved the overall output of a project.

This study is structured into five sections. Section I provides an overview of the PSP methodology, emphasizing its significance in the design and development of software processes and significance of time-logging-based performance monitoring of software engineers. Section II presents a review of the current literature on tools and techniques related to PSP. The proposed framework for performance monitoring along with its validations and study results is described in the third section followed by an elaborated discussion in the fourth section. Finally, the conclusion section summarizes the key findings and outlines potential future research directions.

II. RELATED WORK

Software process improvement tools and techniques help reduce failure rate of IT projects [5]. Most of such techniques also account for issues pertaining to organizational structure and its working procedures. It is a widely accepted fact that projects mainly fail due to time and cost overruns. In this perspective, PSP can help developers detect mistakes at early stages — as the foremost quality objective in PSP is to find and fix defects before the first compilation of a program. PSP pertains to the philosophy of CMM at an individual level to support continuous process improvement [6]. Development and practicing the process performance models are considered as high maturity practices in CMMI. Six Sigma tools are also useful to monitor the performance of processes and can be incorporated in PSP/TSP environment to enhance the overall process efficiency [7]. Monitoring process performance can be accomplished by aligning Six Sigma tools with various PSP tasks. However, this strategy becomes complex due to Six Sigma's diverse data requirements across different processes. Therefore, it is important to carefully select appropriate Six Sigma tools for defect control, such as KANO analysis, scatter plots, and cause-and-effect diagrams. Control theory can aid in identifying critical factors that influence the evolution of PSP [8]. However, it requires the evaluation model to function as a PSP software development prototyping model, encompassing planning of schedules, tasks, reviews, and iterative code development.

Automatic time tracking of the PC activities has become a necessity for organizations to effectively monitor the hours spent by their employees on various tasks. The “PSP Time Logger” by DonationCoder.com is a freeware utility to

record the time spent on development tasks. It allows users to tag various activities through user-defined labels, such as code reviews, documenting requirements, and team meetings. Moreover, several other web-based and cloud-based tools that support time tracking and screenshot recording are commercially available. For instance, [9] introduced AppDOSI, a web-based application for PSP that enables software engineers to control and track their performance. AppDOSI is made up of three elements, that is, software development repository, software development collection and tracking, and software development expertise. Software development repository contains personal data, team data, and organizational data, while management and monitoring data is stored in software development collection monitoring. Analysis data is depicted in a range of formats in the software development expertise component of AppDOSI. Some of the famous tools in this category of software are Editorial Manager® and ProduXion Manager® from Aries Systems Corporation etc. Though most of the available tools allow editable labels to track all the phases of SDLC; the majority of such tools are designed for small businesses.

Researchers in [4] and [6] and studied challenges faced by students in the adaptation of PSP methodology in contrast to the ordinary software development techniques. The methodologies of top-down and bottom-up approaches for process enhancement are addressed in [7]. Top-down approaches involve enhancing software processes from the organizational level down to the individual level. Whereas, bottom-up approaches follow the reverse order, starting from the individual level and progressing upwards. A PSP supporting tool Jasmine [10] facilitates automatic planning, data collection, tracking, and analysis. It utilizes an experience repository to store and disseminate time logs, defect records, and schedule planning information. In this study, [11] developed a method for extracting exact software process activities based on the association between events and activities. The software process activity classier for PSP is proposed to create event-activity mapping relationships from software development event streams, to reveal activity attributes, and to relate the activity to the original SVN log. The proposed method extracts activity from the SVN log based on semantic features and incorporates a novel technique based on a naive Bayes technique to dynamically connect event activities.

The matter of acceptance of PSP within the software engineering domain has consistently been a significant concern [12]. [13] explored factors influencing PSP practices, highlighting the need for an automated tool to support PSP adoption within the software community. Expert Visualization Agent (EVA) [14] has the potential to aid in the development of an automated and adaptable PSP tool. EVA can establish connectivity to the web via Ajax and PHP. It comprises different agents, such as agents related to task, search, and interface that collaborate to provide software engineers with the ability to conduct self-assessment of performance. Accurately calculating the time allocated to each activity is crucial for PSP users, as it facilitates process improvement. To achieve this objective, speech recognition technology can be utilized to log the time allocated to software development activities [15]. Nevertheless, converting the time log generated through speech recognition may pose challenges, as the system may not accurately discern the

specific tasks initiated. Software inspections are the most effective review techniques as these can be performed in the initial stages of the development cycle. Processing the activity time is an overhead for the PSP users. [15] extracted PSP time-log from speech recognition sensor and suggested that heuristic-based trees can be used to generate alternative solutions for time-log. Mouse clicks have also been employed for behavioural biometrics, for instance, researcher in [16] performed mouse activity analysis for user verification. [3] emphasized the influence of code reviews and design reviews on software quality, which can directly affect both the cost and time of the software development process. Therefore, an important consideration in this regard is to track the time allocated to daily activities. Automated and sensor-based task time collection methodologies [17] may serve this purpose. However, the practical application of such an approach requires continuous operation of a speech sensor in the background to record the start and end of activities. By using a PSP tool, the information gathered not only facilitates defect resolution but also enables the assessment of developers' efficiency. Another study [18] proposed incorporating multi-agents into PSP-automated tools for enhanced flexibility and privacy during the process of PSP data collection. A survey report by [19] identified twenty performance prediction and measurement approaches for component-based software systems and assessed their applicability. Substandard software quality may lead towards project failure and financial losses. The responsibility to ensure the quality of software products rests with software engineers and project managers. One key feature of PSP and TSP is the self-directed management approach, which supports software engineers to plan, track, and monitor their own tasks and projects. However, PSP's self-directed management approach would be useless without effective monitoring of the productivity of the development teams besides the progress being made on the development project. This necessitates the support of an automatic performance monitoring tool so that both software engineers and project managers can periodically evaluate their performance without any hassle. The study proposed using a sensor for Integrated Development Environments (IDE), such as Eclipse, Emacs, and Ant etc. The activity time spent can be collected by measuring the time spent on doing the development in the IDEs. In order to collect this time, a sensor can be embedded in the IDE itself which should record the bugs and other errors generated by the IDE along with their time.

Researchers in [20] reviewed data analysis methods used for software process improvement and provided insights into best practices and challenges. However, their study lacked specific guidelines for practical implementation of data analysis techniques. Moreover, they did not address the integration of data analysis with other process improvement methodologies. In their study, [21] discussed innovations in system, software, and service process improvement and highlighted emerging trends and areas for further research. However, researchers in [21] did not explicitly focus on individual developer productivity improvement. [22] proposed a framework to facilitate specifically the adoption of DevOps in software development. They emphasized collaboration, automation, and continuous improvement. However, their proposed framework did not fully address the unique challenges of DevOps adoption in different organizational contexts. [23] explored how

balanced scorecard can enhance software process improvement. In their study, they demonstrated practical implementation of their scorecard method in a small organization. The case study of [23] did not generalize well to larger software organizations. Moreover, it did not delve into the specific challenges faced by small software organizations during the process of improvement. [24] introduced a novel case search method based on activity factors. Their objective was to enhance process improvement efforts by identifying relevant cases. However, the proposed case search method of [24] did not cover all relevant factors affecting software process improvement. [25] presented a theoretical mapping that examined the evolution of software process improvement over the last decade. They identified research gaps and suggested future directions. However, theoretical mapping may lack practical implementation guidelines. Moreover, it does not directly address the role of individual developers in the overall process of improvement.

To overcome these limitations, the proposed PSP framework offers specific guidelines for defect reduction and productivity improvement. It emphasizes individual developer performance monitoring and task management. Moreover, the proposed framework enables automatic scheduling and activity listing for individual developers.

III. PROPOSED FRAMEWORK

This section presents the framework proposed by the current study. The main goal of the proposed framework was to help developers produce quality products within the budget and specified time frame. The proposed framework (Figure 1) may help professionals understand the flow of time spent on the development activities to improve their productivity, efficiency, and quality of work.

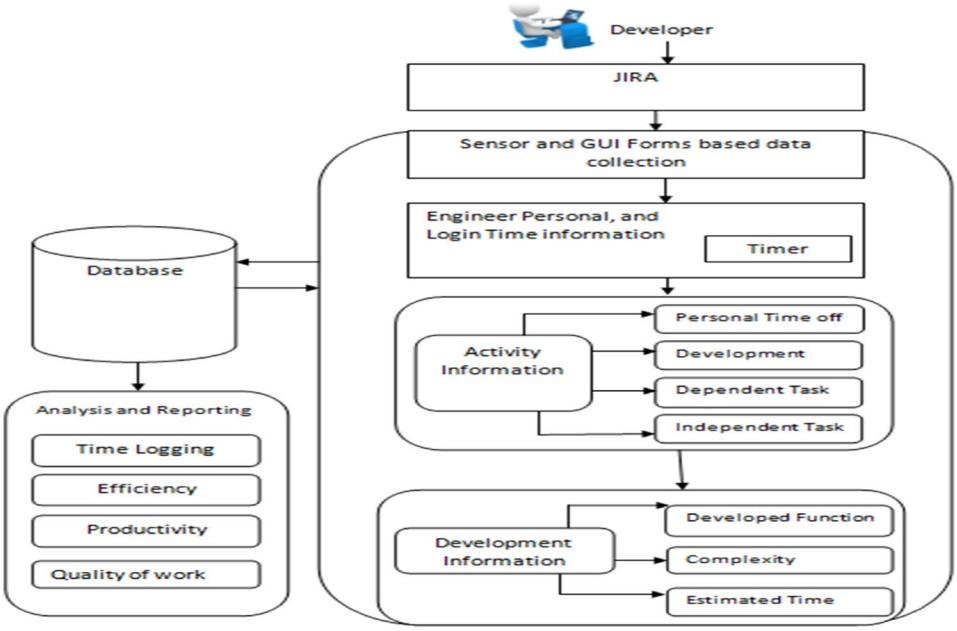

FIGURE 1. Performance monitoring framework for PSP

JIRA is a proprietary product which is used for project planning, management, and tracking applications. In the framework proposed by this study, it was used to assign tasks to the developers. Usually, managers assign tasks to the team members and also mention the estimated time of its completion. In the proposed framework, whenever a developer accomplishes the assigned task, he/she would automatically get another task from JIRA.

Once a developer is assigned a task, then he/she will have to log every activity defined in the proposed framework. For this purpose, the framework proposed in this study used Graphical User Interface **(**GUI) forms to obtain information related to development activities. These activities are organized into four categories: personal time off, development time, dependent task time, and independent task time. This time-logging data is used to monitor developers' time management. Developers would log development details in terms of how many functions they have developed along with the complexity of these functions and their estimated completion time. This information is stored into a database for measuring quality of work, efficiency, and productivity of the developers.

To address the issue of performance monitoring of software engineers and developers, the study proposed a time-logging based framework that is capable not only of monitoring the time spent by a software engineer but also verifies his/her availability at the workplace. The methodology is designed so that if a software engineer does not interact with the IDE for a certain period, it stops logging and generates an alert. To accomplish this technique, the study integrated an Application Programming Interface (API) with the IDE of the software development tool. This API functions similarly to a standard keylogger that automatically collects and measures the time spent on planning, software designing, development, code reviewing, testing, and documentation tasks within the IDE's milieu. However, an unambiguous criterion is required to gauge the performance of developers. For instance, if one developer spent one hour and produced 100 Lines of Code (LOC) with no defects, and another developer spent 30 minutes and also produced 100 LOC with 10 defects. In such a scenario, the key question is to determine who performed better.

The proposed methodology consists of six predefined activity types (planning, designing, development, reviewing, testing, and documentation), three types of user groups (software engineers, team leads and managers), and three skill levels for software engineers (novice, moderate, and expert). However, the project manager can include additional activity types, user groups, and skill levels. In this proposed framework, the managers and team leaders would be able to define tasks for software engineers and may also receive their periodic performance reports for the assigned tasks accordingly. Project-related tasks or activities can be assigned to the software engineers as per the Work Breakdown Structure (WBS) created for the project, or on a daily/weekly basis. However, each task needs to be duly supplied with the expected time duration to complete it. For instance, a manager can list down specific development tasks for the developer 'A' on a particular day; define list of documentation activities to be performed by the software engineer 'B' during a specific time frame; mark out testing activities to be undertaken by the software tester 'C' for the stipulated time and so on.

The proposed model in this study has a provision to generate a time-based performance report after the lapse of specified time intervals set by the manager. The time-based performance report accounts for all the tasks assigned to different software engineers. In case a manager sets the report generation timer to an hour, a performance report will be generated every hour thereafter which contains information pertaining to the total lines of codes/documentation, number of keys pressed, and number of mouse clicks recorded from the specified software engineer's IDE during the last one hour. The performance report would also draw automatic analysis of the activities performed by a software engineer against the predefined minimum threshold level corresponding to his/her skill level (i.e., novice, moderate or expert). If the performance of a software engineer falls below the specified minimum threshold values, then the proposed model would generate an alert to this effect.

The study proposed gathering only the mouse clicks and keystrokes made by the software engineer during project-related tasks. This would help avoid unnecessary collection of several types of logs and snapshots. The idea is to integrate a key logger utility with the API of the proposed framework and make it available in the form of a plug-in that runs in the background on computer systems. The API stores details of the keystrokes in a log file and automatically calculates character and word frequency, as well as maintains counts of the LOC typed in by the software engineer. The API also records the count of deletes and backspace keys. After the lapse of specified time set by the manager, the API would perform a comparative analysis of the keystrokes, words, and LOC count against their corresponding pre-stored threshold values in the lookup tables. For this purpose, lookup tables are linked to the API to carry out comparative analysis.

The proposed methodology entails performing two types of patterns matching keystrokes and keywords specific to a particular development environment. For this purpose, two types of lookup tables or data dictionaries were used: expected word-count lookup table and expected keystroke-count lookup table. The expected keystroke-count lookup table stores the expected count of different keys which are typed in by a software engineer in a specific time span. The expected word-count lookup table contains names of standard functions, constants and keywords of the programming language(s), and tools being used for the development. The specific terminologies and acronyms indigenous to the particular software project are also required to be stored into the data dictionary.

In the first phase, the analysis module matches the keystrokes and words logged in by the API with the corresponding contents of lookup tables. The lookup tables help evaluate whether the work performed by a software engineer relates to project-related activities or something else. For this purpose, an expected yield count is assigned to each word in the data dictionary based on the average values calculated for these words for a specific time duration of the past project-related activities. However, the initial expected yield count for the words can be set based on the previous project or the overall count recorded for these words during the last month.

Nevertheless, the expected word count needs to be adjusted in accordance with the timer settings for generating performance report; thereby, the term 'average expected word count' is used. For instance, if a

specific keyword is observed to have appeared 300 times in the development activities performed by a software engineer during the last 20 working days; then, by taking on average eight working hours per day, the expected average value of that data dictionary word would be 875 per hour. The data dictionary can be periodically updated so that it can commensurate precisely with the current activities being undertaken on a software development project. Furthermore, the average expected value for each data dictionary word can be calculated for three distinct categories based on the software engineer's current level of expertise, that is, expert, moderate or novice.

However, the total count of all the words would also be matched against the overall average count of the words stored in the data dictionary as it is very likely that, depending on the nature of the work, many of the words found in the data dictionary would not have been typed in by the software engineer during the stipulated time period. In other words, the keywords whose reported count is more than the expected average count defined in the data dictionary would compensate for those keywords whose reported count is less than the expected average count.

Likewise, the count of each keystroke pressed by the software engineer during the stipulated time period is matched with the corresponding minimum threshold value set for that keystroke. It helps find the difference between the actual and the expected count for each character (i.e., keyboard key). The difference would be a negative value if the actual count of a specific character falls below the expected count for that character and vice versa. All the resultant values thus obtained are then aggregated and the sum of the delete and backspace keys is subtracted from it to generate the overall total weighted difference. The methodology also keeps records of the spread of the keystrokes to eliminate the situation where software engineers might press some keys randomly. If the difference of the keystroke analysis results in a negative value or considerably a large positive value, then an alarm will be generated. The former case indicates that output of the software engineer was below the expected level. Whereas, the latter situation indicates that the software engineer has simply tried to demonstrate superior performance by fooling the system by typing those keys whose weighted score was higher. To provide cushion for accidentally or arbitrarily striking certain keystrokes by the software engineer, the methodology offers necessary provisions so that a manager can assign an error margin value which could be up to 10% to normalize the predictable keystroke count differences. If 50% of keys listed in the expected keystroke-count lookup table fall below the amount set for their corresponding minimum threshold value, then it can be assumed that the software engineer was not on his/her seat or was busy in doing some other activity not related to the project. A sample of expected values for different keystrokes for three distinct levels of the software engineer's skill level is provided in Table I.

TABLE I
SAMPLE KEYS LIST

Keystrokes	Experienced Developer	Trained Developer	Novice Developer
A	34	30	22
B	19	15	10
(	40	34	30
*	38	31	24
Z	16	13	9

As mentioned earlier, a comparison was also drawn based on count of the platform

specific keywords (i.e., keywords specific to certain development tool e.g., Java, C#, VB etc.) against the data dictionary words. This information is obtained from the key log file maintained by the framework. If the words contained within the log file are related to specific reserved terms of a programming language (or development tool), then their frequency of occurrence is compared against the expected count as defined by the manager in the expected word-count data dictionary. If 50% of keywords individually commensurate with the pre-assigned keywords count and the total word count in the key log file also matches the required number of words to be entered by the software engineer, the system would infer that the developer is engaged in development work, and no alert will be triggered. Table II displays Java keywords' sample along with their corresponding expected values for word-count.

TABLE II
SAMPLE KEYWORDS LIST FOR JAVA

Keywords	Experienced Developer	Trained Developer	Novice Developer
if	24	20	18
for	17	15	11
while	20	14	10
class	28	21	14
int	8	6	5

Additionally, a third type of comparison is based on checking the LOC. The manager would also determine the LOC value based on the anticipated number of lines to be typed by the software engineer within a specific timeframe, such as one hour. During comparison, it would aid in identifying differences between LOC typed by the software engineer and the satisfactory level of LOC established by the manager. Comparison of the total keystrokes, words, and the actual code would be performed concurrently in one go. If the typed LOC count lies within the anticipated count range, as specified by the manager, the development process would be considered to align with the established plan. Another test employed to evaluate the developer's performance involves logging mouse clicks. The proposed framework also retains distinct details regarding the total mouse clicks within the IDE environment.

If the comparison of keystrokes, keywords, and LOC values are not in accordance with the previously mentioned criteria, then a popup message would be generated for both the manager and the software engineer. In addition, the project manager would also receive a performance report for that session for necessary probe, and the software engineer will receive a dialogue box message requiring his/her availability on the seat. Whenever a warning message is generated, the previously mentioned details will also be logged for future record.

There is a provision in the proposed framework to log the "off time" in case the software engineer has to go out for a personal work, or "time-out" in case he/she has to attend the project meeting or any other project-related task which cannot be captured through IDE. However, such atypical log entries would be required to be either entered or verified by the manager. The "off time" and "time-out" entries would be sent to the manager for approval. The current proposed framework aimed to offer a solution that enables both managers and software engineers to collaboratively review assigned tasks and access performance reports.

IV. EXPERIMENT AND ANALYSIS

To validate the model, the study applied PSP level 1 activities to a small team of

developers. PSP level 1 focuses on estimating and evaluating professional performance. Firstly, the current activities of the professionals were captured to understand how tasks are progressing. In the first level, the amount of time spent on each assigned task was measured. A proforma was designed to log the developers' activities which consisted of two parts. The first part was header which contained individual's basic information (name, designation, department, and project name). Whereas, the second part was log area to enter information of the task-related activities (activity description, activity type, start time, time spent on the activity). This format contained the following information. A sample of this proforma is shown in Table III.

TABLE III
PROFORMA FOR ACTIVITY TYPES

Personal Time-off	Any activity which is not related to office or project, such as tea visits etc.
On job	Activities related to official tasks

The sample proforma for activity types and sub-activity types is provided in Table IV and Table V, respectively.

TABLE IV
PROFORMA FOR SUB-ACTIVITY TYPES

Task-oriented	Time spent on development tasks
Dependent task	Any task-related work, such as client interaction, meetings, paperwork, calling clients, discussion with senior officers etc.
Independent task	Any activity which is not related to assigned tasks, such as official commitments including some special work assigned from senior management, training, attending conferences etc.

In this study, forty working hours per week policy were assumed. Based on this assumption, the activities of the individuals were recorded on a daily basis. At the end of the first week, project-wise activity logs of the individuals were collected and evaluated. The study estimated the percentage of time each individual spent on productive tasks. Then, the study focused on the second level of PSP which is concerned with planning and measurements for professional productivity, efficiency, and quality of work. For this purpose, certain levels of complexities were defined related to the development tasks as shown in Table VI.

TABLE V
LEVELS OF COMPLEXITIES RELATED TO THE DEVELOPMENT TASKS

Complexity	LoC (Line of Code)
1	1 – 20
2	21 - 50
3	51 - 100
4	101 – 200
5	201 – 300
6	301 – 400
7	401 – 500
8	Above 500

Following metrics were used to calculate efficiency and productivity of the individuals.

$$Efficiency = \frac{estimated\ time}{actual\ time\ taken} \times 100 \quad (1)$$

If efficiency is greater than or equal to 100, then work is done on or before time.

$$Productivity = \left\{ \sum \frac{Developed\ function \times complexity}{Development\ task\ time} \right\} \times 100 \quad (2)$$

Based on the statistical data collected for one week, the study obtained the following average time utilization by the developers as shown in Figure 2.

The overall summary for a team of three developers is shown in Figure 3.

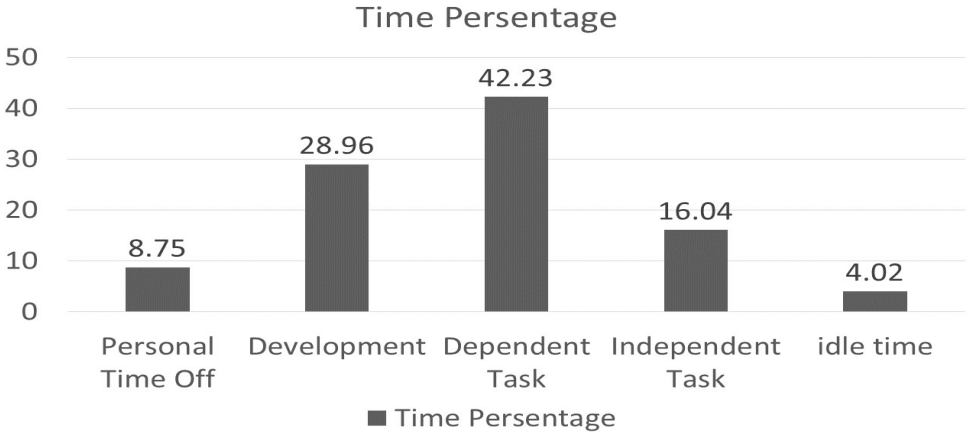

FIGURE 2. Time utilization of an individual developer

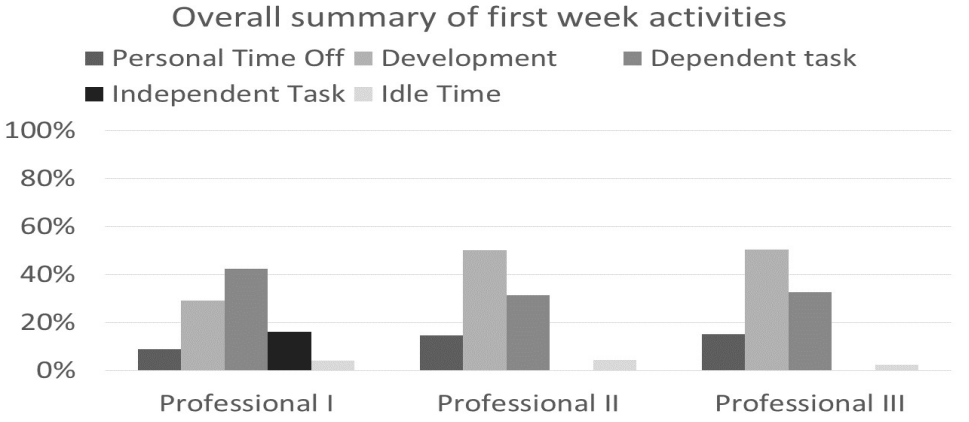

FIGURE 3. Overall summary of first week activities

The study observed the following interesting information from the results collected:

- Availability was not 100%.
- The dependent task ratio was high.
- Development tasks need more concentration as it was merely 29% of the total office time.

To overcome these issues, following specific guidelines were made to increase efficiency of the developers which were implemented for the rest of the duration of

the project. Furthermore, these guidelines helped better manage the project, and the overall productivity of the development team considerably improved.

- Availability time should be 100%, that is, the developers should work 8 hours per day.
- Personal off time should be maximum 10%.
- The development time ratio should be at least 50%.
- The dependent task ratio, such as meetings, trainings, and attending conferences should not exceed 30% of the overall time.
- The independent task time ratio should not be greater than 10%.

V. DISCUSSION

The study developed a prototype model in Java and stored the lookup tables in MySQL database. The model was evaluated for a web-based development project executed by a software house. The project team consisted of a manager, two team leaders, two DBAs, five developers, two software testers, two support staff members, and one technical writer. The model was evaluated by storing arbitrary expected count values for keystrokes and Java keywords based on ingenuity and past experience of the project manager of a similar software project. However, initially, the expected count values were being updated on a weekly basis for the first month by extracting the original count values of the project-related activities stored in the log files. After the lapse of four weeks, the expected count values stored in the lookup tables became more realistic and the performance monitoring software was moved from test stage to production stage.

The current study aimed to solve the problem of time-log processing, daily schedule tracking, and automatic LOC counting. The proposed technique addressed the challenge of the absence of procedures for monitoring the performance of software engineers, especially within the PSP environment. In any PSP tool, there has been a lack of comprehensive attention to performance monitoring. While some performance monitoring activities have been previously proposed, none of them have been integrated with the Integrated Development Environment (IDE). The earlier performance monitoring tools require to be installed separately and capture logs or snapshots of the development process at different intervals of time. Similarly, in certain previous non-PSP techniques, desktop monitoring of developer windows has been previously implemented. In certain techniques, a time-based snapshot of the computer screen is captured. However, the snapshot concept consumes a significant amount of space on the hard disk, even for a single server or monitoring unit. As snapshots are captured at short intervals, analyzing such a large volume of screenshots can become a tedious task. This presents a significant drawback of such techniques, as the manager is required to review the entirety of the snapshots to monitor the performance of a software engineer. However, we believe that a summarized report may suffice for this purpose. Moreover, it is plausible that the developer may be busy with work when the snapshot is taken but it could be possible that he/she would have switched back to the assigned task a few moments before when the snapshot was taken. Conversely, this scenario could also unfold in the opposite manner. However, in both cases, the outcome of such monitoring is inadequate. In the first scenario, the developer may

inadvertently evade monitoring, while in the second scenario, they may face undue scrutiny. In a desktop sharing method, the project manager is required to continuously monitor the developers' screens. Generating performance reports in such methodologies always presents a significant challenge.

The proposed time-logging-based performance monitoring solution incurs minimal overhead in terms of Central Processing Unit (CPU) and disk storage, which is typical for any log-generating tool. Moreover, the proposed technique alleviates any stress on the software engineer during work, as it operates in the background, and alerts are only triggered if the software engineer's performance falls below the expected level. Conversely, in monitoring methodologies involving CCTV or capturing snapshots of the screen, developers often experience psychological stress and pressure while performing their tasks. Furthermore, the proposed technique is not labor-intensive, as the automated tool autonomously analyzes the statistics gathered from the software engineer's computer and generates alerts only when their performance falls below the predefined threshold. The managers can easily see the current statuses of the projects, for instance, percentage of the work done; the time spent on writing code, and count of LOC in the source code. Nonetheless, the proposed technique is quite simple to implement and offers a cost-effective solution. Furthermore, the proposed technique can also be used for the entire development team. The proposed methodology can also be used to find traces of software malfunctions and to conduct digital forensic analysis in the event of misuse of a system. The methodology can also be tailored to accommodate defect logs to ensure better quality PSP products.

A. CONCLUSION AND FUTURE WORK

PSP designates that an organized, controlled, and measured PSP can provide direction and feedback needed to help engineers improve their personal performances. The experiments in this study show that developers can easily improve their capabilities for size and effort estimation, defect density, and productivity. This study aimed to elucidate the concepts of PSP-related techniques, practices, and tools, while also introducing a novel performance monitoring framework tailored for the PSP development environment.

The current study presented a novel technique encompassing multiple methods for automatic data collection and analysis to evaluate the performance of software engineers during certain period of time. This study proposed a performance monitoring model based on PSP and a prototype framework to implement the monitoring model. Furthermore, the study aimed to tackle the challenges associated with continuous recording of time-logging, a feature that other tools typically do not address. In summary, a time-logging-based performance monitoring framework has been introduced that may be seamlessly integrated into a PSP tool. In this proposed model, the manager manually sets the expected number of characters and keywords to be typed by a software engineer within a specified timeframe. However, this approach can be enhanced by automating the process based on specific statistical criteria. For instance, if a multitude of performance scenarios are available from previous projects, a statistical analysis can be conducted to determine the recommended values for the number of characters, words, and LOC which are expected to be entered within a

certain interval of time. This approach would facilitate determination of the suitable range of values for each character and keyword, tailored to the varying skill levels of the software engineers.

The framework presented in this study can help software engineers to practice PSP processes effectively and get feedback on their performance in a timely manner. In addition, the incorporation of framework features would not only help increase the usability of the PSP tool but may also help both the developer and the manager to evaluate their performance and productivity.

CONFLICT OF INTEREST

The authors of the manuscript have no financial or non-financial conflict of interest in the subject matter or materials discussed in this manuscript.

DATA AVAILABILITY STATEMENT

Data supporting the findings of this study will be made available by the corresponding author upon request.

FUNDING DETAILS

No funding has been received for this research.

REFERENCES

[1] W. S. Humphrey, *Introduction to the Personal Software Process*. Addison-Wesley Professional, 1996.

[2] W. S. Humphrey, *Introduction to the team software process (sm)*. Addison-Wesley Professional, 2000.

[3] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on PSP data," *IEEE Trans. Software Eng.*, vol. 35, no. 4, pp. 534–550, 2009, doi: https://doi.org/10.1109/TSE.2009.27.

[4] S. K. Lisack, "The personal software process in the classroom: Student reactions (an experience report)," in *30th Conf. Software Eng. Edu. Train.*, Austin, TX, USA, 2000, pp. 169–175, doi: https://doi.org/10.1109/CSEE.2000.827035.

[5] I. Etxaniz, "Software Project Improvement through Personal Software Process in a R&D Center," in *EUROCON 2007-Int Conf Comput Tool*, Warsaw, Poland, 2007, pp. 413–418, doi: https://doi.org/10.1109/EURCON.2007.4400502.

[6] R. F. Grove, "Using the personal software process to motivate good programming practices," in *Proc. 6th Annual Conf. Teach. Comput. The 3rd Annual Conf. Integr. Technol. Comput. Sci. Edu. Chang. Delivery Comput. Sci. Edu.*, Ireland, 1998, pp. 98–101, doi: https://doi.org/10.1145/282991.283046.

[7] Y. Park, H. Park, H. Choi, and J. Baik, "A study on the application of six sigma tools to PSP/TSP for process improvement," in *5th IEEE/ACIS Int. Conf. Comput. Info. Sci. 1st IEEE/ACIS Int Workshop Component-Based Software Eng. Software Arch. Reuse (ICIS-COMSAR'06)*, Honolulu, HI, USA, 2006, pp. 174–179, doi: https://doi.org/10.1109/ICIS-COMSAR.2006.13.

[8] Y. Yingying, Z. Xianzhong, and W. Wang, "A perspective of PSP

modeling based on control theory," in *2010 Int. Conf. Network Sensing Control*, Chicago, IL, USA, pp. 342–345, doi: https://doi.org/10.1109/ICNSC.2010.5461508.

[9] N. Denwattana, A. Saengsai, and E. Charoenchaimonkon, "AppDOSI: An application for analyzing and monitoring the personal software process," in *2019 4th Int. Conf. Info. Technol.*, Bangkok, Thailand, 2019, pp. 280–283, doi: https://doi.org/10.1109/INCIT.2019.8911993.

[10] H. Shin, H. J. Choi, and J. Baik, "Jasmine: a PSP supporting tool," in *International Conference on Software Process*, Berlin, Heidelberg, 2007, pp. 73–83, doi: https://doi.org/10.1007/978-3-540-72426-1_7.

[11] R. Zhu, Y. Dai, T. Li, Z. Ma, M. Zheng, Y. Tang, et al., "Automatic real-time mining software process activities from SVN logs using a naive Bayes classifier," *IEEE Access*, vol. 7, pp. 146403–146415, Oct. 2019, doi: https://doi.org/10.1109/ACCESS.2019.2945608.

[12] D. Gotterbarn, "Cleanroom, PSP, and the software development impact statement: Developing the right attitude," in *Software Eng. Edu. Train. Conf.*, IEEE Computer Society, 1999, p. 80.

[13] M. H. N. M. Nasir, A. A. Norman, and N. H. Hassan, "Towards a flexible tool for supporting data collection & analysis in personal software process (PSP)," *WSEAS Transactions on Information Science and Applications*, vol. 5, no. 6, pp. 1067–1076, 2008.

[14] H. Hassan, M. H. N. M. Nasir, and S. S. M. Fauzi, "Incorporating software agents in automated personal software process (PSP) tools," in *2009 9th Int. Symp. Commun. Info. Technol.*, Icheon, Korea (South), 2009, pp. 976–981, doi: https://doi.org/10.1109/ISCIT.2009.5340991.

[15] A. Ibrahim and H. J. Choi, "An approach for PSP time log processing," in *Proc. 2008 3rd Int. Conf. Convergence Hybrid Inf. Technol.,* vol. 2, 2008, pp. 676–679, doi: https://doi.org/10.1109/ICCIT.2008.333.

[16] T. Wesolowski, M. Palys, and P. Kudlacik, "Computer user verification based on mouse activity analysis," in *New Trends Intell. Info. Database Syst.*, Springer, 2015, pp. 61–70, doi: https://doi.org/10.1007/978-3-319-16211-9_7.

[17] A. Ibrahim and H. J. Choi, "A Framework for analyzing activity time data," in *2008 IEEE Int. Symp. Service-Oriented Syst. Eng.*, Jhongli, Taiwan, 2008, pp. 14–18, doi: https://doi.org/10.1109/SOSE.2008.32.

[18] M. H. N. M. Nasir, S. S. Hamid, M. K. M. Noor, Z. M. Kasirun, and M. K. Othman, "Using agents to improve the usability of the PSP automated tool," *Int. J. Phy. Sci.*, vol. 6, no. 21, pp. 4977–4989, 2011.

[19] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, vol. 67, no. 8, pp. 634–658, Aug. 2010, doi: https://doi.org/10.1016/j.peva.2009.07.007.

[20] J. Mejía, F. Íñiguez, and M. Muñoz, "Data analysis for software process improvement: A systematic literature review," in *Recent Adv. Info. Syst. Technol. WorldCIST 2017. Adv. Intell. Syst. Comput.*, vol. 569, Springer, Cham, 2017, doi: https://doi.org/10.1007/978-3-319-56535-4_5.

[21] M. Biró, R. Colomo-Palacios, and R. Messnarz, "Advances in system, software and service process improvement and innovation," *J. Software Evol. Proc.*, vol. 31, no. 1, 2019, Art. no. 2146, doi: https://doi.org/10.1002/smr.2146.

[22] J. A. V. M. K. Jayakody and W. M. J. I. Wijayanayake, "Process Improvement Framework for DevOps Adoption in Software Development," in *2023 Int. Res. Conf. Smart Comput. Syst. Eng.*, Kelaniya, Sri Lanka, 2023, pp. 1–7, doi: https://doi.org/10.1109/SCSE59836.2023.10214992.

[23] B. Karahodza, E. Avdagić-Golub, and A. Čolaković, "Applying balanced scorecard in software process improvement: a case study of small software organization," in *2023 46th MIPRO ICT Electr. Conven.*, Opatija, Croatia, 2023, pp. 1697–1702, doi: https://doi.org/10.23919/MIPRO57284.2023.10159756.

[24] Y. Konno and H. Ogasawara, "Proposal of a case search method based on software process improvement activity factors," in *2021 10th Int. Cong. Adv. Appl. Info.*, Niigata, Japan, 2021, pp. 737–742, doi: https://doi.org/10.1109/IIAI-AAI53430.2021.00130.

[25] A. AL-Ashmori, P. D. D. Dominic, S. Basri, Q. Al-Tashi, A. Muneer, and E. A. A. Ghaleb, "Software process improvement during the last decade: A Theoretical Mapping and future avenues," in *2021 Int. Congr. Adv. Technol. Eng.*, Yemen, 2021, pp. 1–5, doi: https://doi.org/10.1109/ICOTEN52080.2021.9493426