AN ASSESSMENT OF WEB - BASED COLLABORATION TOOLS TO SUPPORT SOFTWARE DEVELOPMENT USING THE PERSONAL SOFTWARE PROCESS/ TEAM SOFTWARE PROCESS

THESIS

Presented to the Graduate Council
of Texas State University - San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Roie R. Black, B.S., M.S.

San Marcos, Texas
May 2005

To

Cheryl

ACKNOWLEDGEMENTS

I would like to thank Dr. Carol Hazlewood and Dr. Rodion Podorozhny for serving on my thesis committee and for all of their helpful discussions and guidance during the research leading up to this thesis. I would especially like to thank my advisor, Dr. Greg Hall, both for the motivation for the project itself, and for helping keep it on track in the face of my desires to add far too much material to the project.

I want to thank all the students who have experimented with the PyLIT system as it was being developed, especially the students in Dr. Hall's Software Engineering Practicum class in Spring 2005, whose efforts provided an incredible amount of raw data for this study. Their feedback was valuable in my efforts to produce a tool they would enjoy using.

I would also like to thank my wife, Cheryl, for her encouragement in starting down the road to a second master's degree, and for putting up with my long absences while I went to classes, worked on the research project, and wrote the thesis, while trying to teach as much as possible at the same time.

Finally, I would like to thank the faculty of the Computer Science department at Texas State University - San Marcos for their kind support, and for making the classes so much fun. The second pass through graduate school was much more enjoyable than the first!

This manuscript was submitted on May 15, 2005

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

1.1 Introduction

The computer industry has experienced phenomenal growth over the last decade. In 1993, the USAF operated a Cray-2 supercomputer center at Kirtland Air Force Base in New Mexico to support advanced research in space technologies. The centerpiece machine at that facility was capable of sustaining over 1 billion floating point operations a second, placing it in the list of the top 200 fastest machines in the world (Forbes Corporation 2004). Today, our desktop systems are capable of running at nearly that speed. We have found ways to build systems capable of unbelievable computational power by combining hundreds or even thousands of processing elements into a single system. Computers seem capable of solving almost any problem.

But a computer is only as powerful as the software that drives it. And, while the manufacturers of computer hardware are building faster and cheaper machines at a staggering rate, our ability to produce quality software has not improved anywhere near as significantly. So, we have a fundamental problem. Our powerful computer systems are being run by software full of defects, and we are living in a world where patching mistakes in our software is a way of life.

How do we even begin to address this problem?

1.1.1 A Brief History Lesson

At the end of World War II, General Douglas MacArthur asked Dr. W. Edwards Demming and Dr. Joseph Juran to go to Japan and try to revitalize the Japanese economy, which had

been devastated by the war (Jablonski 1992). These two people introduced the idea of using Statistical Quality Control in the management of the companies in Japan. The basic idea was very simple, analyze the processes being followed in an organization, set up systems to measure what was going on, and use statistics to help make decisions on how to improve the processes.

The result of applying these simple concepts was astounding. Japan emerged as a leading manufacturing country, and earned a reputation for quality that actually exceeded that held by the USA. Companies in the USA began to take notice, and also began to apply the same statistical techniques in managing their operations. The concepts and techniques of applying them evolved into the management philosophy known as Total Quality Management (TQM). By the late 1980s TQM was the dominant theory of management in most companies in the USA.

About the same time that TQM appeared, another statistically based management process, known as Six Sigma, was introduced by Motorola (iSixSigma LLC 2005). Six Sigma began as an attempt to find better ways to assess manufacturing processes and it included several new methods for analyzing defect measurements.

Together, TQM and Six Sigma had a huge impact on the way manufacturing companies worked. These management techniques worked their way into many other service-oriented companies where the focus was on making the customers happy. But how did all of this this impact the world of software development?

1.1.2 Improving the Software Development Process

While all of this attention was being given to manufacturing processes, the software development world was paying attention as well. The Department of Defense established the Software Engineering Institute (SEI) at Carnegie Mellon University to address the entire area of developing quality software. Watts Humphrey, a fellow at SEI, decided to try to

apply the statistical techniques to improve the work of single programmers. The results of this effort became known as the Personal Software Process™ (PSP) (Humphrey 1995).

Once this methodology was defined and placed into practice, it became apparent that another component was missing. Managing a team of programmers was another process that could benefit from the same techniques, so Humphrey developed the Team Software Process™ as another part of managing software development activities. Combining these two process management techniques has significantly improved the quality of software development projects, and PSP/TSP has been proven to improve the quality of software developed, and lower the cost to produce that software (Davis and Mullaney 2003).

1.1.3 The Modern Software Development Team

The PSP/TSP methodologies were originally developed to manage teams of software developers located at a single site. While many organizations do develop software at a single location, it is increasingly common for software to developed by teams of programmers in widely scattered locations. In the extreme case, today's Open Source development activities involve programmers from all over the world.

These development teams rarely have an opportunity for face-to-face meetings, so a new generation of tools and techniques are required to manage these teams. PSP/TSP were not intended to provide a management structure for such teams, but can be used as long as there is support for collaboration to gather and distribute information.

1.1.4 Collaboration Tools

With the explosive growth of the Internet over the last decade, many common tools can be used to enable communications between developers within a single organization, or across the globe. Collectively, these tools are called Collaboration Tools, although several tools

we will mention are not specifically labeled as such.

Bo Leuf (Leuf 2002) discusses three basic types of collaboration tools that use the network as their primary communications vehicle. They are:

- E-Mail exchange - including mailing lists
- Shared folder/file access systems
- Interactive content update/access

The most common of all the collaboration tools is simple electronic mail. Using e-mail, developers can transfer program files, provide management data, receive guidance from team leaders and share in group discussions on a topic using mailing lists. The most serious limitation of using this scheme is the informal (unstructured) nature of the communications, and the lack of timeliness in getting the communications accomplished. However, many projects successfully use just email as their primary communications tool.

A variation of email is becoming increasingly popular: Instant Messaging (IM). In these systems, two people can communicate immediately by exchanging text messages in a scrolling text window. What one user types is sent to the second user immediately, and displayed in their message window. They can reply to the first message by repeating the process. These IM tools are finding their way into the lives of many software developers, and are used until communications must be sent and the second party is not sitting at a terminal, in which case, normal e-mail is used.

Shared folder/file access is another simple collaboration concept. Here a common file repository is maintained at some location, and team members access that repository using some simple tool. Many Version Control Systems use this concept to control the source code for a project. In fact, just about any document can be managed in such systems. Some version control systems even address the problems that can occur when two people

work on a single document simultaneously and attempt to merge their work back into the repository.

In the last category of collaboration tools, content is made visible to team members interactively and changes can be made to that content in real time. Again, there are synchronization problems, but such systems allow teams to maintain a collection of related documents that can be accessed by anyone with network access. In fact, one very well known collaboration system, Lotus Notes (Lotus 2005), supports downloading part of a repository to a local off-line workstation and merging work done back into the main repository at a later time.

In some cases, specialized collaboration tools have been developed. These specialized tools maintain project databases where all kinds of project data can be maintained, and team members run an interface application to access the project database as they work. The data they view are as current as possible, and these tools can support developers in widely separated locations with proper network interfaces.

The direction most network based applications today seem to be taking is to use Web-based tools as much as possible and leverage the web server technology to make information available as rapidly as possible. Microsoft has a large development project ongoing to merge many of their standard office tools into collaboration suites by adding collaboration mechanisms to these programs (McCracken)

1.1.5 The Wiki Web Server

In 2002, Bo Leuf and Ward Cunningham published a book describing a new kind of web server (Leuf 2002). They coined the term Wiki to describe their new server. (Wikiwiki is a Hawaiian term meaning fast, or speedy).

The idea of a Wiki was to allow contributors to make changes to any pages found on the server whenever they wished. This was accomplished by creating a greatly simplified

markup language which could be edited in the primitive editing environment offered by the web browsers most people use. They added a simple edit button to every page on the site, which activated a text input form displaying the Wiki markup that produced the page. The user could modify this markup (assuming they learned the correct markup language) and save the results. The Wiki server would then transform the markup into conventional HTML and the new page could be redisplayed.

A key feature of the Wiki is the ability to generate new pages and link them into the structure of the web site in a very simple way. A Wiki-Word is just any sequence of two or more capitalized words run together to form a single identifier. When the server processes any saved page, it searches for wiki-words. The Wiki identifier is assumed to be the name of another page in the system, and the wiki-word itself is transformed into a link to that page. Clicking on that link will either take the viewer to that page (assuming it exists), or open up a new editor window where the page contents can be created. This simple mechanism allows site visitors to add content to the wiki site easily, and cross references are constructed automatically as the visitor refers to other wiki-identifiers in the system.

This amazingly simple concept has produced quite a following, in spite of the obvious risk in allowing anyone to directly edit pages on a web site. In fact, the Wiki concept evolved in another direction, creating a new generation of on-line diaries known as Blogs.

Wiki servers are traditionally implemented as Common Gateway Interface (CGI) scripts that can be run on just about any web server, and Wiki servers have been actively used in the development of several software products, allowing teams of developers to keep track of a project in a free-form format. The Wiki can hold simple project notes, lists of tasks to be completed, or anything the team decides would be useful.

Wiki systems have found application as free-form note taking systems used while conducting research, to present material in a classroom environment, and in other unexpected ways. The author has used a Wiki on a personal laptop as a tool for managing projects for multiple clients in a consulting business.

The potential of the Wiki is just beginning to be tapped by software developers, and it is apparent that Wiki systems could be extended to provide customized support to the development process. It is this central idea that formed the basis for the current research.

1.2 Research Objective

The original goal for this research project was to explore how far a basic Wiki server could be extended to support software development conducted using the PSP/TSP processes. It was envisioned that tools could be added to collect time data, to drive system compilers and track errors, and to accumulate test results. The system could also be used to produce project documentation, and as a vehicle for processing the forms that are a central part of PSP/TSP. It quickly became apparent that this was too big a project to successfully complete in the time allocated to the project. So, the project was scaled back so it could focus on the feasibility of using a Wiki server as a basic PSP/TSP time keeping tool, and as a tool to construct project documentation.

The project discussed here involved developing a specialized Wiki server which runs as a CGI application on a conventional Apache web server. Components were added to the Wiki server to enforce a simplified form of access control so that teams could collaborate, but not interact with other teams using the system. Other additions were made to support producing project documentation, and then timing components were added to derive time data on the work being performed by the individual developers. The tool was then tested in a software engineering class where small teams followed the PSP/TSP process and used the tool to build their documentation. These teams also kept traditional logs of their time using a simple timekeeping tool developed by Dr. Greg Hall, the course instructor (Hall 2004), and the Wiki maintained independent timing data as well. The wiki system was extended to display team and individual time data logged by the timekeeping tool. The research project sought to compare the accuracy and usefulness of the timing data col-

lected by the Wiki and compare that to data collected using a more traditional approach. In this way the present research seeks to validate that using Wiki technology is a very useful addition to the tool-set available to development teams.

CHAPTER 2

2.1 The Personal Software Process

The Personal Software Process™ was developed by Watts Humphrey in 1995, and is documented in his book *A Discipline for Software Engineering* (Humphrey 1995). The PSP was developed to assist individual developers plan their work, manage their time, and apply quality principles to build better software (Humphrey 2000). PSP introduces a process improvement strategy based on the following key actions:

- Defining a quality goal
- Measuring product quality
- Understanding, using and modifying the process
- Measuring results
- Comparing results with goals

It should be readily apparent that PSP depends on good data collection, so developers are trained to keep detailed records of their activities. These records are used throughout the development project and are reported to team managers to assist in managing the project. Unfortunately, in today's world, most individuals fail to accurately track their activities (Davis and Mullaney 2003).

Even worse, they do not see the value of tracking all of the information required, nor do they believe that doing so will help them in any way. Some developers remain suspicious that time tracking is only used in individual performance evaluations and not

as a way to build better products. Some developers attracted by the promise of PSP/TSP have avoided using the techniques due to the lack of adequate data collection tools (Morisio 2000)

However many studies now prove otherwise, and especially in the classroom environment, individuals are learning to become more effective by paying attention not only to what they produce, but what they actually do to produce these products.

2.1.1 Overview of PSP Activities

PSP, in the full implementation, proceeds through a series of levels of activity designed to help train the developer to focus on different aspects of a project. The levels basically look like this (Humphrey 1999):

- PSP0 - Baseline Personal Process
- PSP1 - Personal Project Management
- PSP2 - Personal Quality Management
- PSP3 - Cyclic Development Level

PSP0 - Foundation Level

In the foundation phase, the developer is introduced to the major development phases in the project: planning, design, coding, compiling, testing, and postmortem analysis. There are four forms used to record activities at this level: Time Recording Log, Defect Recording Log, the Project Plan Summary, and a Process Improvement Proposal. The goal at this level is to focus on data collection and analysis to try to learn how to better estimate the time it will take to complete tasks.

At the second level, the developer concentrates on estimating project size and estimating time to complete tasks. Since the primary product is usually software, the traditional measure of lines of code (LOC) would seem to be the logical choice for what to estimate. However, this can be very difficult to do, since LOC can vary widely from one programmer to the next offering little consistency in the numbers produced.

Instead PSP uses the Proxy-Based-Estimate method (PROBE). In this scheme, the developer looks for some other measurable "proxy" that can relate the size of the project to something that can be measured and estimated effectively. A simple example of such a proxy would be the number of objects or classes introduced in an object oriented project.

PSP1 introduces two new forms to the record keeping set: the Task Planning Template and the Schedule planning Template. The Project Plan Summary is also used to record estimates produced in this phase.

In PSP1, statistical methods such as linear regression are used to relate prior size and time estimates to actual data and are used to predict future values. A prediction interval is calculated to determine an expected range around these estimates based on statistical variances. This interval can be used to measure the quality of the estimates.

In estimating time, an earned value method is used. This technique assigns a planned time estimate for tasks based on that task's estimated percentage of total project effort. As tasks are completed, those planned estimates become earned values, and together these two values can be used to estimate rates of progress and completion dates for a project (Hays and Over 1997).

PSP2 - Quality Improvement Level

In PSP2, quality management techniques are introduced. In this phase, the developer seeks to find and remove all defects in the products produced. The standard measurement used to

track this information is yield, defined as the percentage of defects injected before compile time that were removed before compiling. Of course, the ideal situation is one where all defects (100 percent) are removed before compile.

In this phase, developers conduct design reviews and code walkthroughs. A Design Review Checklist and Code Review Checklist are used in these activities. In addition, historical data is used to estimate the number of future defects and future values for expected yield. The review process is designed to teach developers how to examine their work from many different perspectives to find as many defects as possible before compile.

PSP3 - Cyclic Development Level

In any project, a certain amount of administrative overhead is always part of the process. For small projects, this overhead can seriously impact productivity. At the other extreme, large projects can reach a level of complexity where productivity is also limited. PSP3 introduces decomposition techniques where large projects can be broken up into smaller sub-projects whose size can be kept in the range where productivity is maximized.

In this phase, the developer uses forms to track a summary of size estimates, development time, and defects per cycle. They also document issues that arise that may impact future cycles.

2.1.2 PSP Effectiveness

The Personal Software Process has been the subject of many studies to validate it's effectiveness. In particular Davis and Nauman conducted a structured test of 298 developers in a classroom environment. Their results are summarized below (Cannon 1999)

- Effort estimates improved by a factor of 1.75
- Size estimates improved by a factor of 2.5

- Product quality - defects per unit test, improved by a factor of 2.5
- Process quality - defects found before compile, improved by 50 percent
- There were no significant changes in productivity or LOC

While these results are encouraging, there is concern about the quality of the data used to produce these results. Disney conducted experiments to assess the impact of data accuracy on results and found that for a small set of nine developers, working on a total of 89 projects, over 1500 errors were found in recording the basic data collected (Disney 1998). This raises a serious question as to whether the overall effectiveness of PSP is better or worse than reported in actual use. In general the use of PSP is accepted as a good way to improve the development process, but the questions remain about how effective it could be if better data collection techniques could be found.

2.2 TSP Overview

In conventional PSP/TSP activities, the time logs and weekly summaries are sent to team managers who integrate the data collected and conduct team planning and measuring, ultimately resulting in feedback to the individual team members to direct their future work. Humphrey decided that the same basic techniques introduced for individual developers could be applied to managers of teams as well (Humphrey 2000).

The Team Software Process process is designed to help teams of developers better control costs, scheduling and the quality of the work they do. TSP can be used in all phases of traditional software development. TSP depends on individual developer's input data collected using the PSP process.

Watts Humphrey published a book on TSP aimed at students learning to be more effective software engineers (Humphrey 2000). In this book, student team managers learn how to coordinate teams, establish team goals, and track team activities.

The TSP process focuses on keeping teams properly directed, and seeks to maximize the effectiveness of each team member. This is a complex management task, and many organizations develop their own management styles to address the issue, often with widely varying degrees of success. TSP seeks to provide a consistent framework for managing truly effective teams.

Davis and Mullaney (Davis and Mullaney 2003) present an overview of the key elements of TSP. These include:

- Team Communication
- Team Coordination
- Project Tracking
- Risk Analysis
- Goal Setting
- Role Assignment
- Tailored Team Process
- Detailed Balanced Plans

In addressing each of these elements, TSP clearly depends on effective communication methods to keep the teams and team managers working together. In fact, one of the most important management activities in TSP is a weekly meeting of team members. This is clearly not easy when teams are made up of people in widely scattered locations. However, teleconferencing and creative use of the Internet affords an opportunity for communications to occur even in this situation.

2.2.1 Overview of TSP Activities

As with PSP, TSP involves a series of basic levels of activity. These are:

- Team Member Skills Assessments
- Team Building Activities
- Team Management Activities

TSP was designed to help managers build effective teams quickly, and move them to higher levels of productivity and quality in the products they produce. In using TSP, managers break a project into a number of distinct phases, called cycles, with clearly defined products to be produced in each phase (Humphrey 1999).

At the beginning of each cycle, meetings are held to set the stage for the work to follow. In these meetings, managers and team members establish project goals, assign responsibilities, and produce estimates of the time the tasks will take, and the size of the products to be produced. Once the project starts, weekly meetings are held to review PSP data and this information is used help guide individual team members in their efforts to improve their performance.

Team Member Skills Assessments

In developing individual skills, TSP managers focus on establishing a disciplined approach to conducting the project. Individuals are encouraged to work independently, but to stay within boundaries set by the overall team. The team members learn how to use the various PSP logs that help track their activities, and how to use the data they collect to produce their own estimates of future performance. Finally, they learn how to apply quality measurements to better manage their work.

In the second major area, the focus is on team building. Here, reasonable project goals are established and individual tasks assigned to the team members. Team building actually begins as the development cycles are launched to begin a new round of development activities. The tasks to be performed should be challenging enough to teach the individuals that overall project success will only happen when the entire team succeeds.

The cycle launch is accomplished by holding team meetings where all aspects of the project are discussed, and where the team can be motivated to succeed (Davis and Mullaney 2003).

Team Management Activities

Once the cycle is underway, the TSP managers hold weekly meetings to collect data and adjust the activities. Managers use the same basic forms as those used in PSP, but their focus is on making estimates for the entire project, and balancing the work of each individual.

Team management introduces a few new reports which help summarize team results. These reports are usually a redisplay of existing data from different project perspectives, such as component, development phase, task, or week. By examining the project data from many different perspectives, new insights can be formed into how the project is proceeding, and problems can be avoided.

2.2.2 TSP in Practice

Again, several studies of the TSP methodology have been conducted to assess it's effectiveness. Cannon reports results from use of TSP in a project at Teradyne. These results are shown in Table 2.1 (Cannon 1999).

Since a measure of the effectiveness of management activities is the ability to accurately predict how a process will proceed, these numbers are fairly good. Detailed results

	Plan	Actual
Size	110KLOC	90KLOC
Effort	16000 hours	14711 hours
Schedule	77 weeks	71 weeks
Defects in Integration	1 defect/KLOC	.2 defects/KLOC
Defects in System Test	.1 defect/KLOC	.4 defects/KLOC
Defects in Field Trial	0 defect/KLOC	.02 defects/KLOC

Table 2.1: Teradyne TSP Results

Measure	Average
Productivity Improvement	78 percent
Failure Reduction	58 percent
Total Failure Analysis Time Reduction	30 percent

Table 2.2: Summary of TSP Effectiveness

from another study of the effectiveness of TSP from a study by Davis and Mullaney (Davis and Mullaney 2003) is shown in Table 2.2.

2.3 Examining PSP/TSP Interactions

After studying the PSP and TSP systems, it became apparent that there was a uniform way to show how these two processes are integrated into a software development system. This new view of the system is helpful in critically examining how data collection and analysis are used to guide the overall project. In order to present this view, we need to examine how the major players in a PSP/TSP software development project interact.

2.3.1 Identifying the Major PSP/TSP Interactions

In any development project, there are two groups of people who do the real development work. They are the developers and the managers. Developers are those who perform most

of the creative work associated with the project, managers are those who guide the work associated with the project. Some individuals involved in a project may perform both functions, but we will distinguish them by the role they play at any moment.

These two groups do not simply start off developing software. Instead, they follow a set of procedures established to guide how they will function as the development proceeds. These procedures may have been established by the development organization, or they may be defined by outside sources. In the case of projects using PSP/TSP, the general guidelines are defined by those processes.

The interactions between the developers, managers and procedures can be seen as a matrix where the major activity of each component interacts with the major activities of each of the other components. Before trying to describe the interaction matrix further, the major activities must be defined.

2.3.2 Major Activities of Developers

Without considering exactly how individuals are guided in their work, there are four primary types of activities any individual can engage in:

- Creative thinking and research
- Engaging in conversations or meetings
- Creating documents and code
- Performing administrative record-keeping for the project

In all of these activities, written materials should be created, although in some cases that material is quite informal. The creative thinking and research activities often generate simple notes or references. The actual creation of code and documentation obviously produces more formal written products. The administrative activity includes record keeping,

time keeping, and other overhead activities associated with just about all work an individual does. For that reason, administrative activities are not considered as a separate activity, but a part of all activities. PSP subdivides several of these categories into multiple parts, but we will ignore that detail for the moment.

2.3.3 Major Activities of Managers

Managers are responsible for analyzing the performance of the team and assessing the quality of the work that is being accomplished. They review progress on assigned tasks, analyze data produced during the work, and ultimately, assign new tasks to keep the project on track. They also have responsibilities in building cohesive and motivated teams. They control the course of the project by tracking the addition of new work tasks to the project, and supervise quality concerns. In summary, these are the most important activities of the managers of the development project:

- Maintain a list of active tasks
- Evaluate project data
- Assess Team performance and progress
- Control project cycle process
- Administrative overhead activities

Much of the manager's activity involves examining data produced by the developers. This material is in the form of written reports or forms, and is often delivered to the manager in project meetings.

2.3.4 Major Project Activities

In a project using PSP/TSP, the major activities are divided into categories that correspond to the primary phases in the development of a software product. At the very least, the following activities would be included:

- Gathering Requirements
- Designing the System
- Writing Code
- Testing Code
- Support and Maintenance functions

These are clearly the major phases of just about any software development project. Some development teams might introduce variations of this list to accommodate their specific development philosophy.

2.3.5 A Perspective of the Interactions

While it is true that managing a software development project is a very complex process, we can consider that the major product of the management activity is the production of a list of items to produce for the project. Realizing this, we can simplify the manager's activities into the establishment of that list, and other activities to control the quality and timeliness of the work of the team. We can lay out a simplified matrix of the interactions as shown in Figure 2.1.

The item set created by the manager can be organized according to the major phases of the development process. In the figure, the item set is shown as a vertical bar which we can divide into subsets of items associated with each of those development phases. The individual developer conducts work on each of the assigned items. Time will be spent

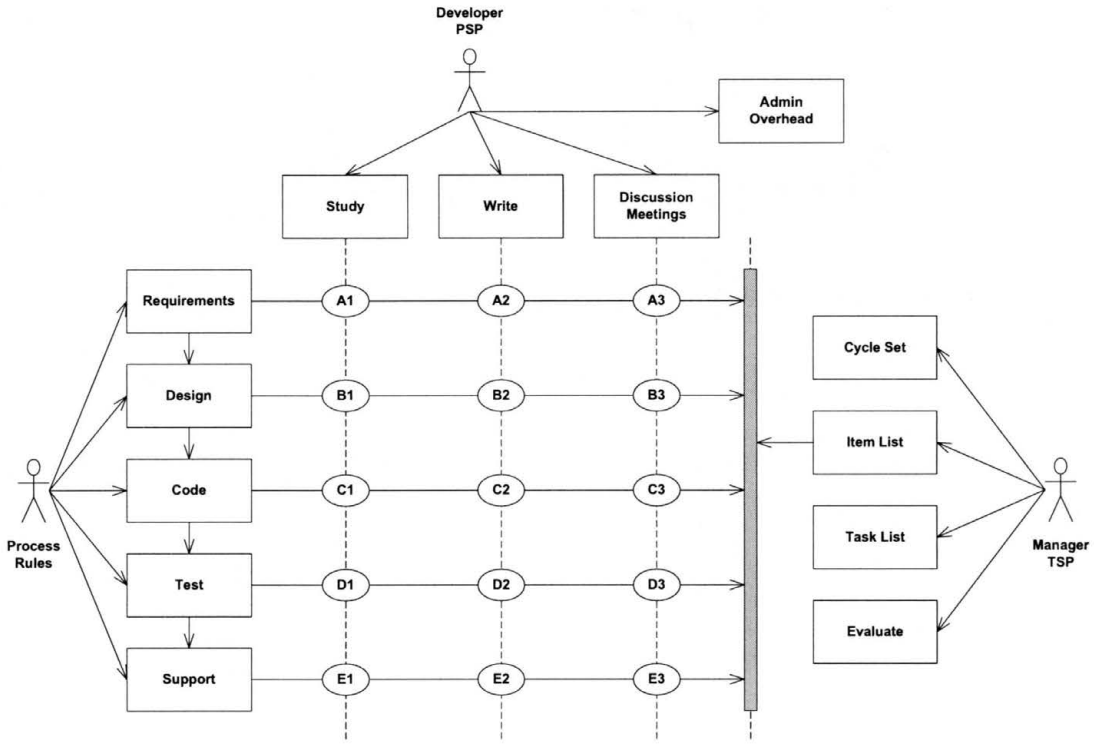

Figure 2.1: PSP/TSP Interactions.

in each of the individual activity areas as the work on an item is accomplished. The lines drawn down from each category of activity intersect with a phase line drawn from the phase to the item bar. At the intersection of each of these lines, there is some specific activity that takes a finite amount of time to accomplish.

At each intersection, there must be two fundamental products produced: a time log of the time spend working at that point, and a concrete written product of some kind that we can quantitatively measure (using the PROBE technique discussed earlier). We can also examine the qualitative measures established for the project by conducting peer reviews or inspections. Controlling the development process depends on collecting accurate useful data at each of these intersection points, and using those data to assess project progress, and to adjust the work as it proceeds to project completion.

We can see that using this matrix can assist us in examining what data we might

wish to collect to assist in managing the project, and what kind of activity we are examining. Since we are considering projects conducted using PSP/TSP, we will examine the data collected in those processes, and use the matrix as a guide during later analysis.

2.3.6 PSP/TSP Data Collection

Data collection begins with the individual developer who tracks all project related activities in an Engineer's Notebook. In theory, the engineer will record time spent on every aspect of the project, a goal seen initially by many as a bit intrusive on their daily work. The notebook typically contains sections for the following items:

- Project Assignments
- Project Notes
- Design Notes
- Time Tracking

The first three of these are normally kept in electronic form, such as word processor document files, project management databases, or spreadsheets. Timekeeping, on the other hand, relies on manual entries in some form of log, often a paper record. Since time records are one of the key components of PSP/TSP, using paper records is a serious problem. There is a significant risk that transcribing these records into a database or spreadsheet will introduce errors. This entire process can be quite time consuming.

PSP Time Logs

The daily log section of the Engineer's Notebook is designed to record details of every activity undertaken by the individual. A typical time tracking entry is shown in Table 2.3:

Most of these entries are obvious. The Interruption entry allows for stopping an activity to engage in an interruption. The Delta Time is defined as the total time for the activity minus the interruption time. The last two columns record the state of the activity (complete or incomplete) and the units of measure for the activity (time in minutes, or pages read).

Weekly Activity Summaries

At the end of each week, the entries from these logs are summarized in a Weekly Activity Summary. In this document, the typical activities an individual engages in become apparent, and the daily amount of time spent on each can be totaled. In addition, this information can be compared to previous weeks activities to look for trends, and to computer rates of production.

2.3.7 Planning Forms

A number of the forms uses in PSP/TSP are used to record planning information. These forms record estimates of future time and product sizes. Most of the material included in these forms is derived by analyzing data recorded by the engineers, and will not be discussed further in this report.

2.4 PSP/TSP Data Flow

An overview of how the data collected in PSP/TSP is used to manage team activities is presented in Humphrey (Humphrey 2000). A simplified version of the process is shown in

Date	Start	Stop	Interruption	Delta Time	Activity	Comments	Completed	Units
2/1	3:58	4:47	3	46	0.2	Read PSP Chapter 2		27 pages
2/11	9:29	9:50		21	0.2	Writing		min

Table 2.3: Time Tracking Log Entry

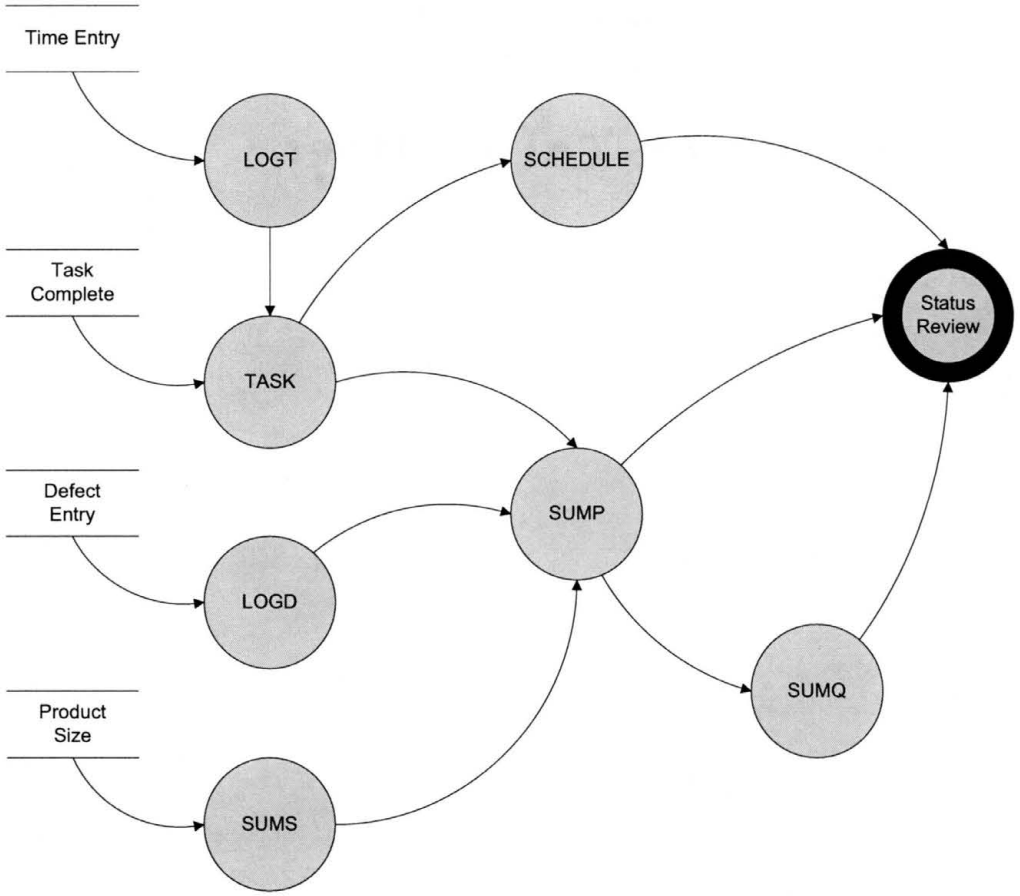

Figure 2.2: PSP/TSP Data Flow.

Figure 2.2.

The engineers produce the basic time data (LOGT), product size data (SUMS), and defect data (LOGD) that drives this data flow. A status report on the state of assigned tasks is recorded on the TASK form as well. This information is used in reviewing the project schedule, and in summarizing the state of the products being produced. A quality assessment is also derived from this information. All of this material is used during status review meetings, where the managers provide feedback to the engineers as a guide for the next weeks activities.

2.5 Summary of PSP/TSP

There is little reason to view the activities of PSP and TSP as an unreasonable added cost to the development process. A large body of evidence, beginning with the introduction of TQM and Six Sigma techniques, and extending into research in the effectiveness of PSP/TSP has proven that using statistical methods to help control the direction of teams can significantly improve the quality of the work performed, and the products produced (Humphrey 1999). Ultimately, the customer satisfaction level increases, which is what we all strive to accomplish.

PSP/TSP can improve the productivity of developers, and the quality of their products. What is needed is research into ways to better accomplish these improvements while minimizing the overhead introduced by the extensive record keeping activities needed in any statistically controlled activity.

2.6 Automating PSP/TSP

The PSP/TSP processes could benefit from some form of automation. However, most attempts to introduce automation into the processes have focused on the recording and statistical processing of PSP/TSP data.

The time log is a critical part of the overall PSP/TSP process. It is a fairly informal document, but it has proven quite adequate for personal use. Traditionally, these logs were kept on paper. Manually logging information on these forms can result in errors. In fact studies of the quality of PSP data have shown that the errors can seriously compromise the effectiveness of the overall process (Disney 1998). For this reason, introducing some form of automation to the process has been explored. These automated tools range from simple spreadsheet or database front end systems to make sure calculations are accurate (Disney 1998), to more effective tools that perform some of the time tracking actions.

The primary source for PSP/TSP training is the Software Engineering Institute, where Humphrey continues to conduct seminars on the processes. A review of the materials offered by SEI shows that primarily, practitioners of the process depend on simple tools such as customized spreadsheets (Software Engineering Institute 2005). Because of the proprietary nature of the training offered by SEI, these tools are only made available to those participating in SEI training.

A fairly extensive search for other tools was conducted as part of this study. Unfortunately, no commercial tools were located, however, several fairly simple tools have been developed as part of software engineering courses introducing the PSP/TSP processes into their programs (Morisio 2000)

An example of such a tool was developed by Dr. Greg Hall at Texas State University for use in his Software Engineering Practicum courses. The tool developed by Dr. Hall is used in the current study (Hall 2004). A similar, but somewhat more extensive tool, the PSP Studio, was developed at East Tennessee State University (Henry 1997).

In some development environments, the collecting of personal data is regulated to a level where using PSP/TSP could even be deemed illegal. An attempt to develop a tool that can record and integrate individual data in a controlled manner was presented by Escall and Morisio (Escala and Morisio 1998).

One very interesting Open Source tool is available for use in PSP/TSP guided development projects. The Dashboard system (http://processdash.sourceforge.net 2001) is designed to provide both time keeping support and assistance in laying out the overall project plan. Dashboard is a web-based tool in that it uses the web browser as it's primary interface. Dashboard is installed as a server application that supports collaboration over the Internet. However it is primarily used as a data recording and analysis system, with little support for other development activities. There were plans to extend it's data collection systems by integrating it with existing development tools. The program was first released in 2001, but does not appear to have been actively supported since 2003.

2.7 Motivation for the Current Research

The basic premise of the current research is that there are better ways to collect the fundamental data that form the basis for most PSP/TSP activities. Current collection techniques are unduly cumbersome and error prone, raising the risk that the potential of the PSP/TSP may be compromised by bad data. The availability of tools to help reduce the overhead of PSP/TSP is very limited. The present work seeks to explore alternative methods that can acquire the required data with little of no burden on the individual developer.

CHAPTER 3

3.1 Research Methodology

The present study focused on the quality of timekeeping data collected in typical PSP/TSP managed projects. The study used data collected during software engineering classes conducted at Texas State University during the Spring 2004 and Spring 2005 semesters. The overall methodology used in the research is discussed in the next few sections.

3.1.1 Project Goals

The goal of this project was to collect sufficient activity time data to assess the effectiveness of three time keeping methods: traditional manual forms, an automated date entry system, and a fully automatic data recording scheme. The present study does not try to validate the PSP/TSP procedures, that has been adequately discussed in the literature. The present research seeks to show that adding fully automated time keeping systems to PSP/TSP can significantly enhance the usefulness of those procedures and improve the quality of software produced.

Collected Data

The specific data collected for this study includes basic time log entries and data on the major products produced during a typical software development project. The time records during the 2004 classes were collected using the standard PSP LOGT paper forms. During 2005, this data was collected by using the PSPtimer application developed by Dr. Hall, which serves as a simple stopwatch that can record the time data directly into a project

database. The program also allows developers to enter manual records to capture events that take place away from the workstation. The program records the data collected in standard LOGT format.

Also during the 2005 classes, detailed timing data were collected automatically by the PyLIT system developed for this study. In this case, the developers used the system to produce the project documentation and timing data were collected during those activities. Due to limitations in the PyLIT system (discussed later), detailed timing data on the actual program code produced during the development project were not collected. However data on the project documentation were included in the research.

During both semesters, the developers worked through two development cycles, each beginning with a requirements collection phase, then proceeding to a design development phase. The actual coding and testing of the software followed, however, since these activities were conducted using conventional development tools on individual workstations, data on these later activities were not collected.

Expected Data Quality Issues

Based on research into projects that have used PSP/TSP mentioned earlier, we can summarize the data collection issues expected to be encountered in using these processes to manage a development project:

- Paper logs - error prone
- Support systems - better, but still subject to errors
- Automated systems - accurate data but possibly incomplete

3.1.2 Overview of Research Tools

In this section, we present a brief overview of the primary tools used in this study.

The PSPtimer application is a simple timekeeping and logging tool developed by Dr. Greg Hall for use in his software engineering classes. The application provides a simple graphical user interface designed to be present on the workstation whenever the developer works on a project. The most important function of the timekeeper is a stopwatch function that can be started, paused, and stopped by the developer as needed during any work session. The stopwatch window is shown in Figure3.1.

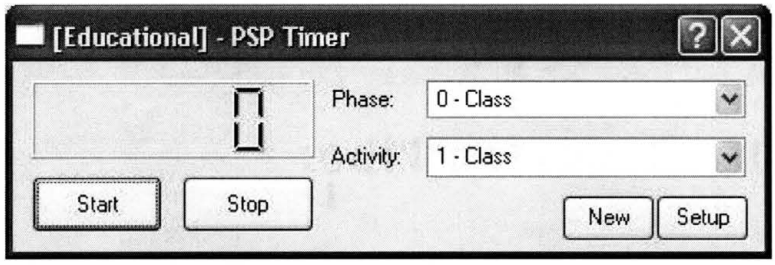

Figure 3.1: PSPtimer Stopwatch Window.

For those work sessions taking place away from the developer's workstation, the PSPtimer also provides a way for timing data to be entered after the work is completed. In this case a different form is used to log the time data. This form in shown in Figure3.2

Both of these forms require the developer to enter basic information about what development phase is being worked on, and to record other data on the amount of work accomplished. The tool automatically records the entered data in a database for further processing.

In the current study, this tool was used unmodified to collect PSP/TSP data using conventional manual recording methods. The advantage of having a system that would record the data into a database was expected to improve the quality of the data collected using traditional methods.

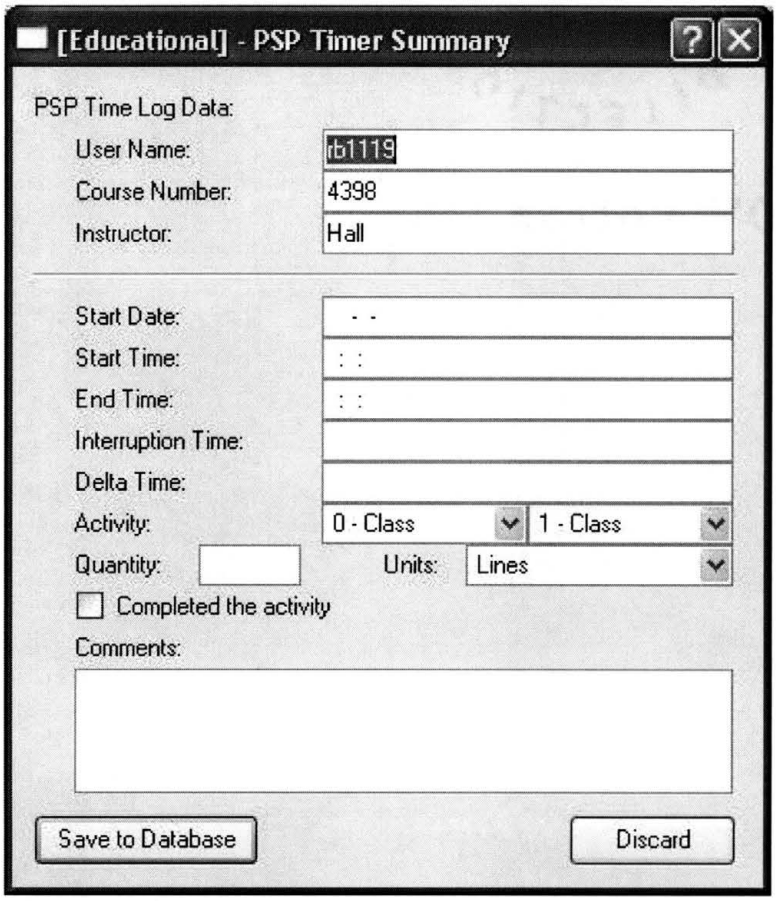

Figure 3.2: PSPtimer Data Entry Window.

The PyLIT Wiki System

The PyLIT system developed for this research is a conventional Wiki system augmented to support the PSP/TSP processes. Specifically, the system provides access controls to manage shared access to project documents, and places a project management structure on top of the normally free-form Wiki system.

Shared access to project documents was implemented using a simple password system, and a template system was developed that establishes a standard outline for the major sections of the Software Requirements Specification and the Software Design Specification documents. The study participants were divided up into teams each of which were provided with access to the template root documents for their documentation. Individual

team members could add to these root documents as desired to document their work. All pages created during the project were recorded in a central study database.

A time keeping logging system was added to the Wiki system to track individual activities while these documents were accessed on the Wiki server. The logging system allowed data to be collected on both reading and writing phases of document production. In addition, data on exactly what changes were made to the documentation by each team member could be collected. The outline for the documents being produced was used to determine what development phase was undergoing work during any session.

As a backup to these data collection methods, PyLIT also recorded detailed logs on it's internal operations, and logs were maintained by the Apache Web Server. These backup systems provided confidence that the data collected during this study would be as complete as possible.

The developers were to use the PyLIT system to produce all project documentation during the Requirements Specification and Design phases of the project. In response to comments by early users of the PyLIT system, a Javascript editor was added to the system so that users unfamiliar with wiki markup could edit documents in a familiar editing environment. Further details on the PyLIT system are available in Appendix A.

3.2 Data Analysis Procedures

Once the data were available, the manual forms were transcribed into a database for further processing. The data from the PSPtimer and PyLIT system were also loaded to the same database. Basic data processing was accomplished using a set of scripts to produce the data tables and graphs shown in this report. A separate script performed the statistical analysis presented as well. A schematic of the data entry process is shown in Figure 3.3.

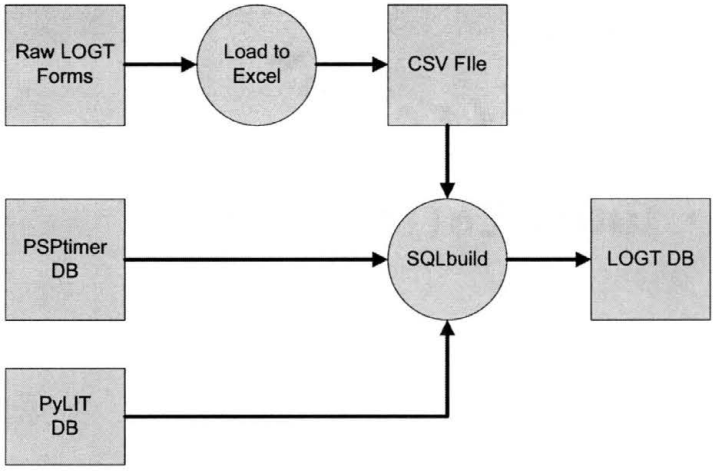

Figure 3.3: PSP/TSP Data Flow.

3.2.1 Assessment of Data Accuracy

Since the goal of this study was to assess the accuracy and completeness of the data collected, a simple analysis of the accuracy of both the manual paper logs, and the data produced using the PSPtimer application was also analyzed. This analysis examined all the data available in the records looking for any errors. Errors that could be encountered ranged from incorrect start and stop times, interruption time errors, and actual work time calculation errors.

The data from the PyLIT system was not subject to manual modifications, and was deemed accurate.

3.2.2 Statistical Analysis

The most important analysis of the study involved examining the data collected to see if it could accurately predict the size of the documents produced during the study. Accordingly, the time data was processed at both the team and individual level using linear regression to attempt to correlate document sizes to time spent during the requirement and design phases

of the project. 34

CHAPTER 4

4.1 Hypothesis

This study was based on the hypothesis that an automated tool could collect PSP/TSP data with accuracy sufficient to greatly reduce, and in some cases eliminate, the need for extensive manual record keeping. The current work did not attempt to cover all aspects of data collection for PSP/TSP, but to demonstrate the effectiveness of a web based collaboration tool in collecting timing data for a subset of the activities of a typical software development team.

4.2 Evaluation of the Basic Logged Data

An initial assessment of the collected data was performed to ensure that there was sufficient data for the study. This assessment looked at the total number of entries logged, and the coverage of the PSP/TSP data collection domain. It also examined the initial data errors detected while recording the test data into a central analysis database.

4.2.1 Summary of Log Entries Collected

The students in both semester's projects produced a significant number of data records. During the 2004 class, all of this data was recorded on paper logs which were transcribed into a database by a purely manual process. The data were checked as much as possible to eliminate transcription errors, but no other data errors were corrected during this process.

Phase	Class	Reading	Meetings	Writing	Comm	Reviews
Class	250	113	17	3	27	0
Reqs	0	16	12	56	35	13
Design	3	8	11	57	14	20
Coding	9	36	29	131	19	9
Testing	1	2	4	15	4	6
Support	4	6	6	17	10	9

Table 4.1: Total Records Logged by Phase/Activity (Manual 2004)

To gain a perspective on the total quantity of data collected a simple analysis of the number of events logged and the distribution of log entries over the duration of the classes was produced. Table 4.1 shows the number of records collected for the 2004 class for each of the development phases and activities. Figure 4.1 shows the distribution of these entries over the entire semester.

The noticeable sequence of events starting at about 2PM coincides with the class time associated with this semester's activities. The time spent in class was logged as part of the process of training the students to record all activities associated with the development project.

For the 2005 semester, the PSPtimer was used to collect raw PSP/TSP data. This program could be used as a stopwatch during activities conducted on the student's workstation, and could also be used to record data from activities conducted away from the computer. The raw records logged into the central data base were also plotted to show overall activity during the semester. Table 4.2 shows the total log entries collected by this tool. Figure 4.2 shows the distribution of those events.

Finally, the data recorded by the PyLIT tool were also analyzed. In this case only the requirements and design phases of the development work were captured. Table 4.3 shows entries for the class phase of the development. In fact no class time was logged by PyLIT, so this field was used to record entries that could not be associated with a single

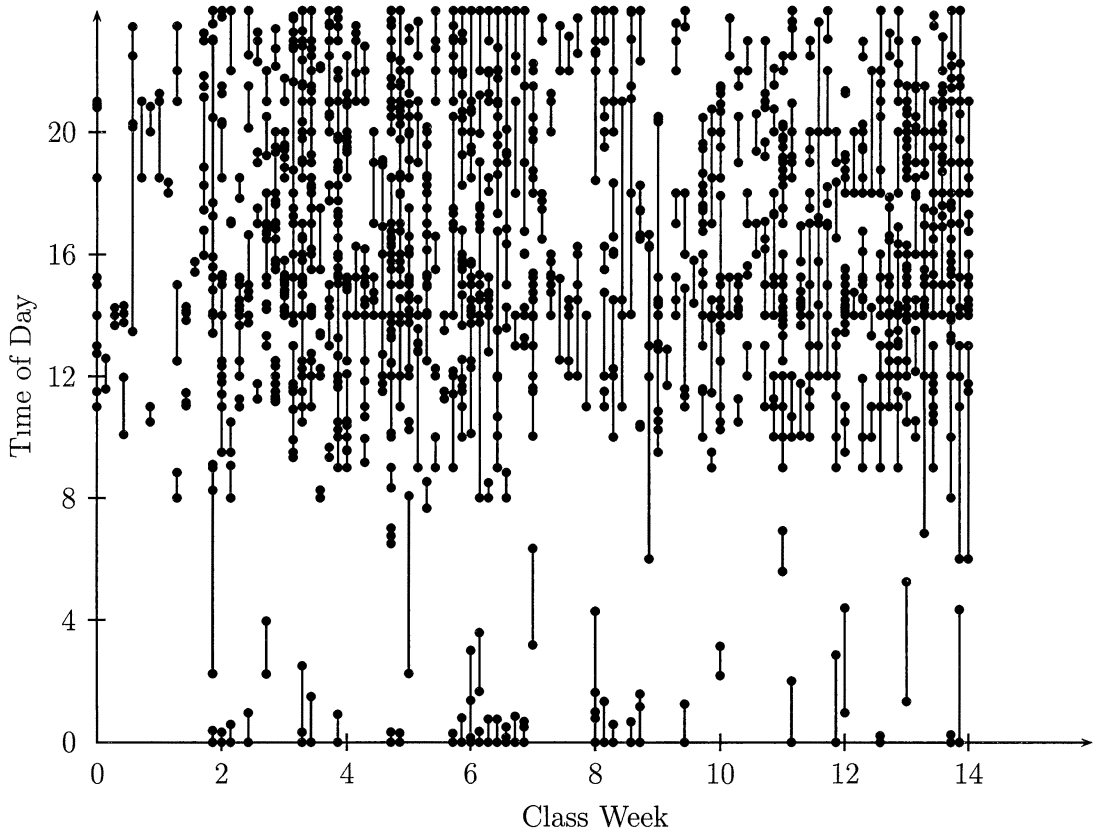

Figure 4.1: 2004 PSP/TSP Records

user. These log events correspond to server activity before a student logged in to the PyLIT system.

The unclassified writing entries are the result of log entry patterns that could not be correctly assessed. These events occurred when a user started an edit session, then failed to save the page opened for editing and then decided to look at other pages. The log entry patterns were too complex to correctly classify in these cases. These records were ignored in further data processing but are included in these summary tables as an indication of the number of records with unresolved problems remaining in the system.

The distribution of PyLIT log entries over the semester is shown in Figure 4.3

The PyLIT program produced a log record on every page request sent to the server. The start time of each request was recorded in the database. A post–processing pass over

Phase	Class	Reading	Meetings	Writing	Comm	Reviews
Class	32	30	10	0	15	0
Reqs	4	6	30	74	18	18
Design	1	10	19	81	11	14
Coding	18	21	15	175	16	8
Testing	1	0	1	19	4	7
Support	0	0	2	2	7	0

Table 4.2: Total Records Logged by Phase/Activity (PSPtimer 2005)

Phase	Class	Reading	Meetings	Writing	Comm	Reviews
Class	0	1082	0	264	0	0
Reqs	0	650	0	329	0	0
Design	0	1255	0	1252	0	0
Coding	0	0	0	0	0	0
Testing	0	0	0	0	0	0
Support	0	0	0	0	0	0

Table 4.3: Total Records Logged by Phase/Activity (PyLIT 2005)

the data determined the duration of the activity. The time and character of the next recorded event for an individual student was processed to determine when the previously recorded event finished.

As mentioned earlier, this scheme occasionally produced activity records that could not be properly attributed to a single student, so they were marked as unclassified.

There were over 11,000 individual log entries recorded by PyLIT, The post processing pass reduced the number of entries to 5000. However, many of those entries are still short enough to collapse into a single point in the figure. Since every request sent to the server could be recorded, it was possible to distinguish between read requests and write requests at a very fine level of detail. In fact, the only time recorded as writing was that time actually spent editing a page on the server.

At first glance, Figure 4.1 seems to indicate that the PyLIT tool is not capturing as much data as necessary. The events are very short, but this is simply an indication of

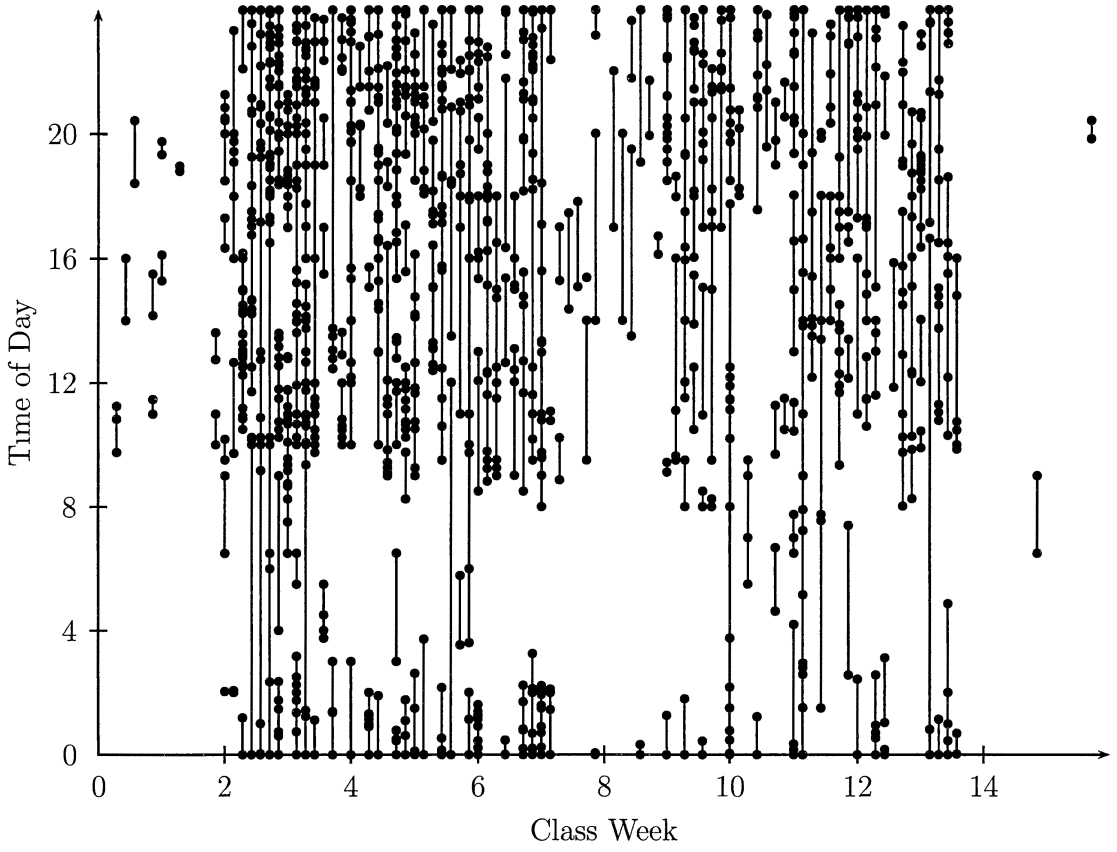

Figure 4.2: 2005 PSPtimer Records

how fine-grained the tool is in collecting data. The sufficiency of the data can only be determined by comparing the data collected by all tools to make sure we are looking at the same basic time data.

PyLIT was capturing data only during reading and writing activities during the requirements and design phases. These activities were expected to consume significant time in the development projects. If we redraw the first two figures to show only this data, the PyLIT results are much more encouraging.

Figure 4.4 shows the relevant time data from the manual data forms from 2004. The distinct lack of data shown in this figure is quite interesting. One would think that the time spent reading and writing in the requirement and design phases would be much greater

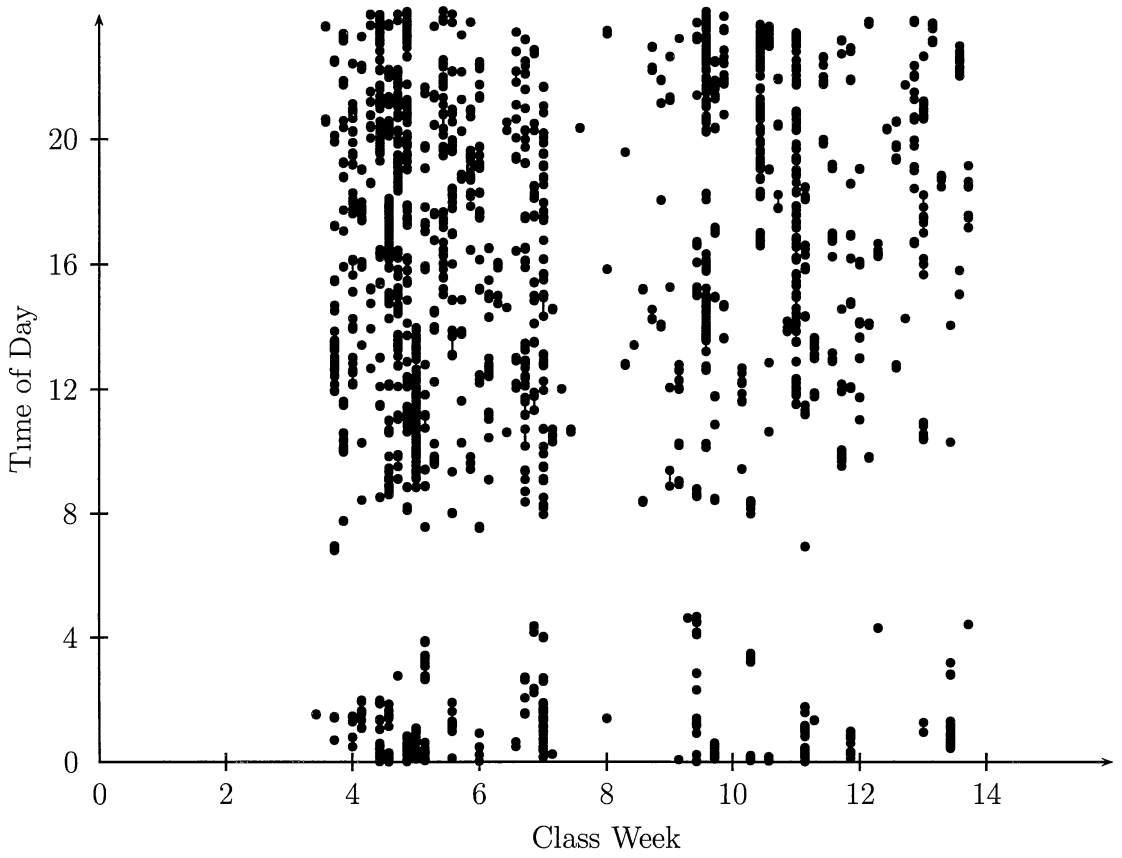

Figure 4.3: 2005 PyLIT Records

than the data indicates. Further analysis of this data will show where the student time was actually accumulated.

Similarly, Figure 4.5 shows the same basic times recorded by the PSPtimer program for the 2005 classes. Once again, the total time recorded is significantly lower than would be expected.

A summary of the total time accumulated by the students during both semesters is shown in Table 4.4. When the data from all three data sets included just those records associated with reading and writing during the design and development phases the total number of hours logged is similar. The difference in the total hours logged by the PSPtimer and PyLIT programs can be attributed to reading activities conducted away from the worksta-

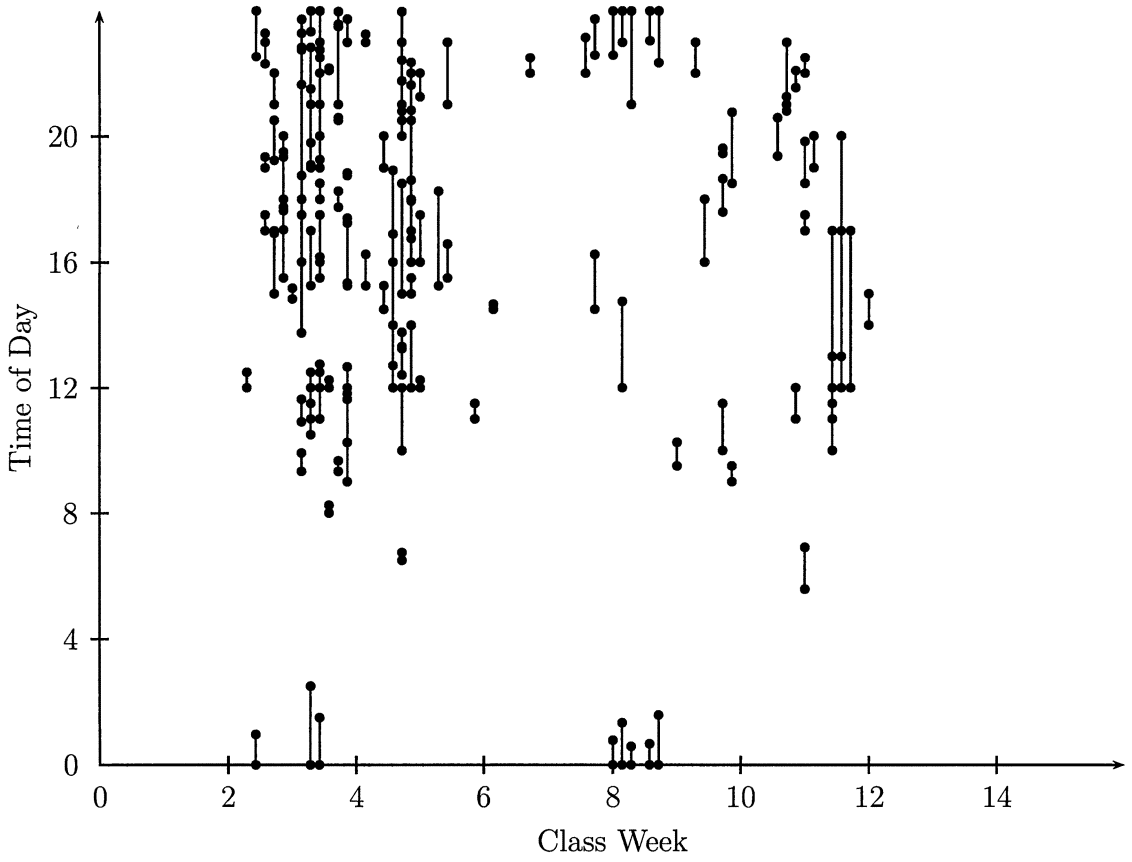

Figure 4.4: 2004 Manual Records

tion. All writing events should have been captured, but we will examine this issue in a later part of this discussion. Overall, these results are again encouraging and indicate that the automated tool is capable of collecting data with no burden on the user.

4.2.2 Analysis of Data Errors Detected

Both the manual PSP/TSPdata forms collected during the 2004 semester and the data collected by the PSPtimer application during the 2005 semester showed data errors.

The data from the manual data collection was very difficult to work with. The forms were hard to read, which resulted in transcription errors when attempting to load the data into a database system. Further, the recorded data was often inconsistent. Dates

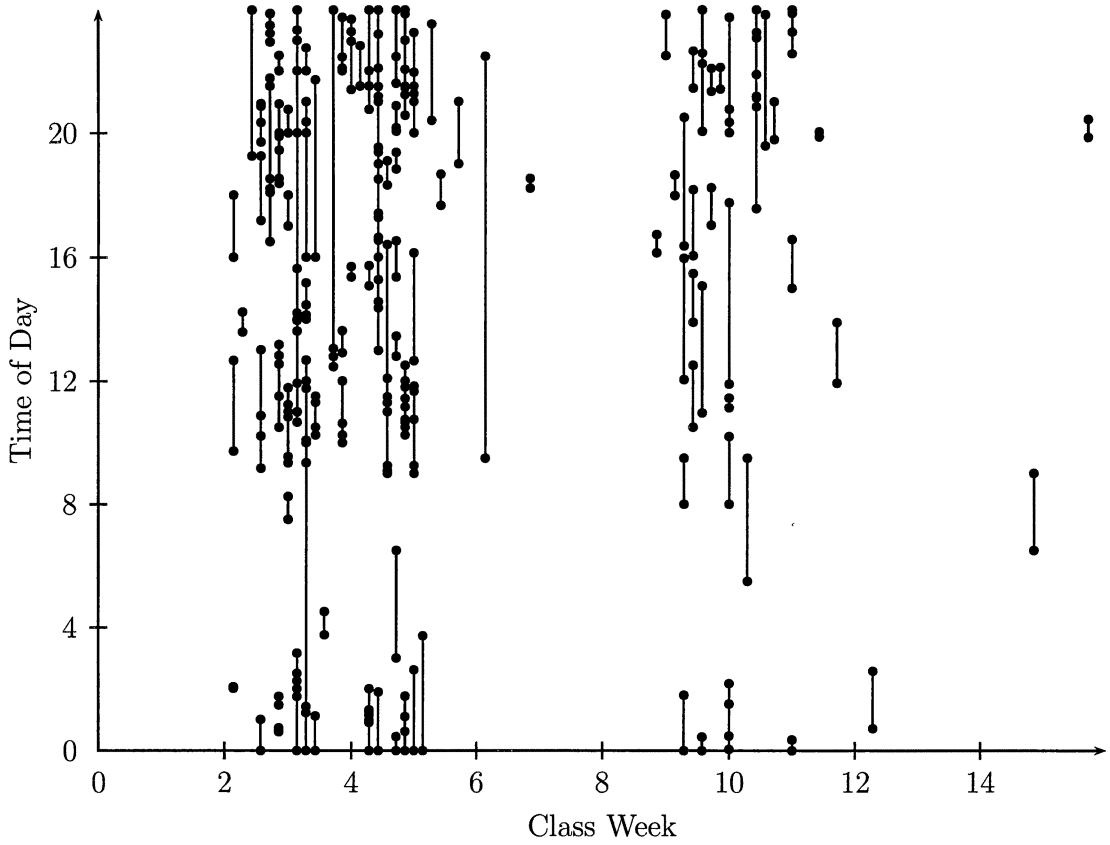

Figure 4.5: 2005 PSPtimer Records

were found to be incorrect, start and stop times transposed and inconsistent time units were detected. As much as possible, these inconsistencies were detected and corrected before further time analysis was conducted, but the records were still marked to indicate they were in error. A quick calculation of the total event time for each log entry was then performed and compared to the actual number recorded by the student. These delta times were often found to differ by large amounts. It appears that asking an engineer to do the arithmetic needed to fill in this part of the form is a big source of data errors in manual PSP logging systems.

In further processing of the data for this study, it was assumed that the start, stop and interruption times were valid. If the calculated delta time differed from the entered value, the calculated value was used.

Source	Total Time	Read/Write	R/W Req/Design
Manual	1336.8	753.1	206.5
PSPtimer	1461.0	975.3	335.9
PyLIT	115.1	115.1	115.1

Table 4.4: Summary of Total Hours Logged

The PSPtimer data also contained many errors. This was surprising, given the simplicity of the user interface. The problems arise because the system allows manual entry of events that take place away from a workstation. In the current implementation of the program, the data are not validated before recording the entry in the database. As a result, dates were again wrong, or missing in many cases. And, once again, delta calculations were often off.

When the data were processed it was found that more errors were present in the PSPtimer data than existed in the manual records. The fact that the data were loaded directly into the database made it worthwhile using this system, but it is clear that the program needs to be modified to provide better error checking.

The PyLIT data was error free, but it suffers from missing some of the key information needed to get truly accurate data on time spent. Specifically, there is no way to know if an engineer has left the terminal and stopped working on the project with no direct input from that user. Instead, the logs show periods of inactivity, which can be interpreted many ways. For this study, a lack of activity of 15 minutes of more is considered an interruption. This value can be adjusted to see the impact of waiting a longer time for activity, but in the absence of better guidance this number was chosen as a baseline value.

A summary of the number of data errors detected by each collection system is presented in Table 4.5.

Source	Total Errors	Read/Write	R/W Req/Design
Manual	126	72	21
PSPtimer	157	91	40
PyLIT	0	0	0

Table 4.5: Summary of Data Errors Detected

4.3 Summary of WEEK Data

The timing data collected during PSP/TSPactivities is used by team managers to assess the productivity and overall progress of the project. Normally, these data are integrated by the managers and used in weekly team meetings where development issues are discussed. Therefore, the next analysis to be performed on the study data will look at that data over the course of each week's work. Since the projects assigned to the teams during the semesters were broken up into two complete cycles, we will look at the weekly data during each cycle as well. For this analysis, we will restrict the data analyzed to just that covered by all three systems.

It should be noted that the students were working on their projects with several defined deadlines established for the semester. Each team worked two eight week cycles, which produced versions of the final software for their project. Each cycle of the project involved producing a Software Requirements Specification (SRS) and a Software Design Specification (SDS) before coding was to take place. The SRS documentation was due at the end of the fourth week of class and the SDS documentation was due at the end of the sixth week. Accordingly, we should expect to see activity near those dates, and less activity after each due date.

Source	W1	W2	W3	W4	W5	W6	W7	W8
Manual	0.0	0.0	21.8	59.5	50.0	9.1	0.7	4.1
PSPtimer	0.0	0.0	37.8	87.5	92.3	26.4	13.3	0.0
PyLIT	0.0	0.0	0.0	4.2	18.7	12.8	9.2	5.6

Table 4.6: Total Time Logged for Cycle 1

Source	W9	W10	W11	W12	W13	W14	W15	W16
Manual	15.7	9.2	5.2	30.2	1.0	0.0	0.0	0.0
PSPtimer	0.6	31.8	35.2	6.1	1.9	0.0	2.5	0.6
PyLIT	0.6	12.1	4.7	10.4	1.7	4.9	0.0	0.0

Table 4.7: Total Time Logged for Cycle 2

4.3.1 Weekly Team Total Times

The first analysis conducted looks at the distribution of effort by each team over the course of the eight weeks assigned to each project cycle. Tables 4.6 and 4.7 show the total time logged by each team for the three systems. The totals in these tables exclude unclassified records, so the total hours presented here is lower than the total times recorded in Table 4.4.

In examining the data in these tables, it appears that PyLITis not recording as many hours of work as is actually being performed. Specifically, the PSPtimer data indicated many more hours per week for the teams. However, the significance of these differences can only be seen if we split the times into read and write times. These results are shown in Tables 4.8, 4.9, 4.10 and 4.11.

Source	W1	W2	W3	W4	W5	W6	W7	W8
Manual	0.0	0.0	3.9	6.3	0.0	1.1	0.5	0.0
PSPtimer	0.0	0.0	1.0	12.0	9.2	0.0	0.0	0.0
PyLIT	0.0	0.0	0.0	2.4	5.7	3.4	4.0	1.1

Table 4.8: Total Read Time Logged for Cycle 1

Source	W9	W10	W11	W12	W13	W14	W15	W16
Manual	5.2	0.0	0.0	1.5	0.0	0.0	0.0	0.0
PSPtimer	0.0	3.9	0.0	0.0	0.0	0.0	0.0	0.0
PyLIT	0.1	6.4	3.0	6.0	0.8	2.7	0.0	0.0

Table 4.9: Total Read Time Logged for Cycle 2

Source	W1	W2	W3	W4	W5	W6	W7	W8
Manual	0.0	0.0	17.9	53.2	50.0	8.0	0.2	4.1
PSPtimer	0.0	0.0	36.8	75.6	83.1	26.4	13.3	0.0
PyLIT	0.0	0.0	0.0	1.8	13.0	9.4	5.2	4.5

Table 4.10: Total Write Time Logged for Cycle 1

Source	W9	W10	W11	W12	W13	W14	W15	W16
Manual	10.5	9.2	5.2	28.7	1.0	0.0	0.0	0.0
PSPtimer	0.6	27.8	35.2	6.1	1.9	0.0	2.5	0.6
PyLIT	0.5	5.7	1.8	4.4	0.9	2.2	0.0	0.0

Table 4.11: Total Write Time Logged for Cycle 2

A close examination of the data in these tables reveals discrepancies that raise serious questions about the validity of timing logged by the students using techniques that allow manual data entry. The differences in time spend reading recorded by PSPtimer and PyLIT could be explained by allowing for reading activities conducted away from the workstation and later entered into the PSPtimer system. However, during 2005 cycle 2 , these was practically no reading activity logged, which is a highly unlikely result. PyLIT was recording activity during this same period, however, it is likely that the students did not consider the use of that system properly and did not log any time spend working with the system as reading.

A more serious question arises in examining the write time data for 2005. Since the students were supposed to use only the PyLIT system to produce their documentation, it is unlikely that the large times recorded as writing in the PSPtimer system are totally accurate. Some students decided to use other tools to produce portions of the documentation and later upload that material to the PyLIT system. Some of the material incorporated into the documentation was produced by graphical tools like Microsoft Visio. The figures produced by these tools were later uploaded to the PyLIT system, again with some loss of accurate timing data. Even considering these two situations, the recorded times still seem unusually high.

A literature search did not produce any studies of PSP/TSP data collection where a fully automated tool was available for comparison. The findings of the present study certainly raise questions about the validity of manually collected data, at least where developers are just starting with the record keeping process. It is likely that these students were not tracking their activities with the care really needed, a fact that is certain to change as they see more value in the overall data collection process.

It is also apparent from these tables that the write time for cycle 2 is consistently lower in general than for cycle 1 in all systems tested. This is certainly due to the fact that the cycle 2 development documents were simple extensions of cycle 1 documents with

minor changes.

We do see increased activity in all systems around the project due dates mentioned earlier, as expected. Therefore, it appears that all three systems are recording data that accurately reflects the individual work effort.

4.3.2 Regression Analysis of PSP/TSP Time Data

In attempting to find a liner model that can be used to predict measures of a development project, we are effectively attempting to prove that the future project data can be predicted by our model. If the model is perfect, the model will predict all data points exactly. However, in the real world, the data points we are working with will differ from those predicted by our model.

We need a way to assess the differences between our model and actual data. The most common measure of this difference is a Root Mean Squared Error (RSME) (Draper and Smith 1998). In calculating this value, we are looking for small RSME values, indicating that our model matches the data fairly closely. Large values may indicate that our model is not accurately representing the data, and perhaps we need to look at other models.

In our analysis, we are trying to find a simple measure that a manager can use to assess the overall progress of a project. The data for this prediction will naturally be noisy, since the developers have noisy schedules and will not be applying a consistent level of effort toward completing the project.

So, what kind of trend are we seeking? PSP/TSP typically uses linear regression to seek a trend in the time data that can be used as a measure of progress. In the face of clearly noisy data, the justification for using this approach comes from the realization that software development processes will always have variable complexity levels in the components produced, and varying skill levels in the developers themselves. As long as the variations between the model used for evaluating the process lie within controlled bounds,

the process is said to be statistically controlled (Janiszewski and George 2004)

It has been shown that data collected in PSP/TSP development projects cannot be interpreted incorrectly if only team totals are assessed. Individuals need to be guided, not just teams, so examining individual variations may be more useful that simply looking at team totals. This fact argues for developing better methods for collecting the basic data used in the processes.

Since in normal use the PSP/TSP data are processed to predict both productivity and product size values, a regression analysis was performed on the time data collected in this study. Two primary results are reported here. In the first, the total records available on each day of each cycle were evaluated to produce daily linear regression values which were them plotted to show daily variations. The values at the end of each week were then extracted to show the values that team managers would normally use to produce their assessments in a traditional PSP/TSP development.

The liner regression equations are summarize here Hardy and Bryman (2004):

$$S_x = \sum x_i$$

$$S_y = \sum y_i$$

$$\bar{x} = \frac{S_x}{N}$$

$$\bar{y} = \frac{S_y}{N}$$

$$S_{x2} = \sum x_i^2$$

$$S_{xx} = \sum (x_i - \bar{x})^2$$

$$S_{yy} = \sum (y_i - \bar{y})^2$$

$$S_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y})$$

$$m = S_{xy}/S_{xx}$$

$$b = \bar{y} - m\bar{x}$$

Traditional statistical variables can be computed from these equations as well. Standard deviation of the residuals is given by:

$$S_r = \sqrt{\frac{S_{yy} - m^2 S_{xx}}{N-2}}$$

The standard deviation of the intercept is

$$S_b = \sqrt{\frac{1}{N - \frac{Sx^2}{S_{x2}}}}$$

The standard deviation of the slope is

$$S_m = S_y / \sqrt{S_{xx}}$$

The correlation coefficient is:

$$r^2 = \frac{S_{xy}^2}{S_{xx} S_{yy}}$$

The analysis of the manual PSP/TSP time records from 2004 are shown in Figure 4.6. The line shows the times predicted by the linear model based on the time data available at each point. A manager would like to be able to use the linear model to predict development time in the future. If the regression analysis shows a fairly flat trend over the course of the study, then the values predicted early in the study will be a good indicator of the overall results. From Figure 4.6 it is apparent that the predictions very early in the development process are not very stable and will not give a good final estimate. However, as the project progresses, the numbers do tend to stabilize and the linear analysis will give good results. In the one week period around week eight, we see a gap in data. During this time the students took a one week break from school, and little activity was recorded.

The data used in this analysis is clearly very noisy, but we are concerned only with looking for trends in the data that can be used to effectively guide the progress of the team. In this context, we can see a trend that is useful.

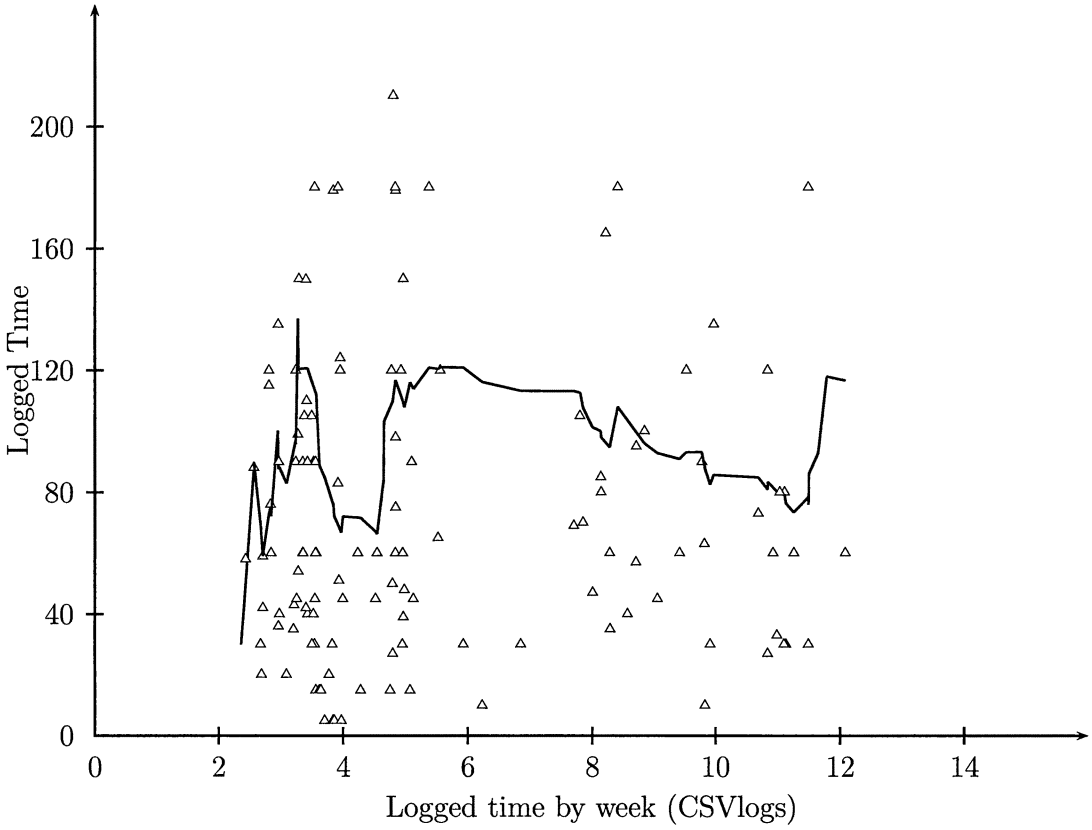

Figure 4.6: 2004 CVSlog Linear Regression Results

Figure 4.7 shows the same results for the 2005 study using data from the PSPtimer application. Once again, we see an unstable prediction in the early weeks of the development process, but the linear model does stabilize over time and again gives good results in later stages of the development process.

Finally, Figure 4.8 shows the analysis with data from PyLIT. In this case, the individual time entries logged much smaller time increments, and we have significantly more data to analyze. In examining this graph, we see a different pattern that needs interpretation to assess the usefulness of PyLIT in predicting development times.

As is the case in the first two results reported, the data recorded in the early phases of the study is not really adequate to give good overall predictions. As the development proceeds, the predictions do get better. In the first eight weeks of the effort, the overall

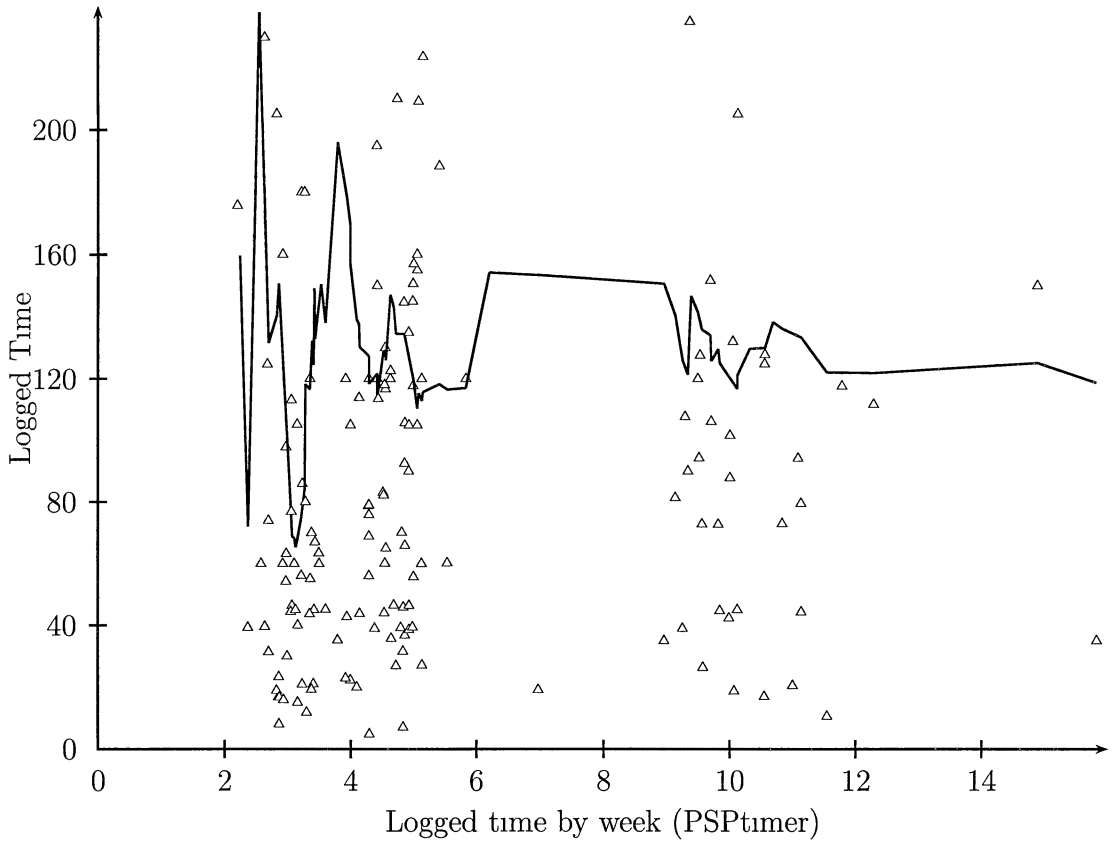

Figure 4.7: 2005 PSPtimer Linear Regression Results

trend is not as flat as in the previous two graphs, and we see a distinct spike in activity toward the eight week mark. This coincides with the end of the cycle 1 activities, when a report on progress was due. At the beginning of Cycle 2, we see another spike in activity as the students continued work, and a final spike toward the end of cycle 2.

All of this behavior would be expected by a team manager, so the regression data could be effectively used in predicting development efforts. PyLIT data seems to be much more closely tied to the real work being performed by the developers. It is likely that manual data recording techniques may suffer from lack of use when pressures to produce results increase on the developers. At this point the PyLIT system will accurately record the activities.

A summary of the weekly statistical data computed by the analysis is shown next.

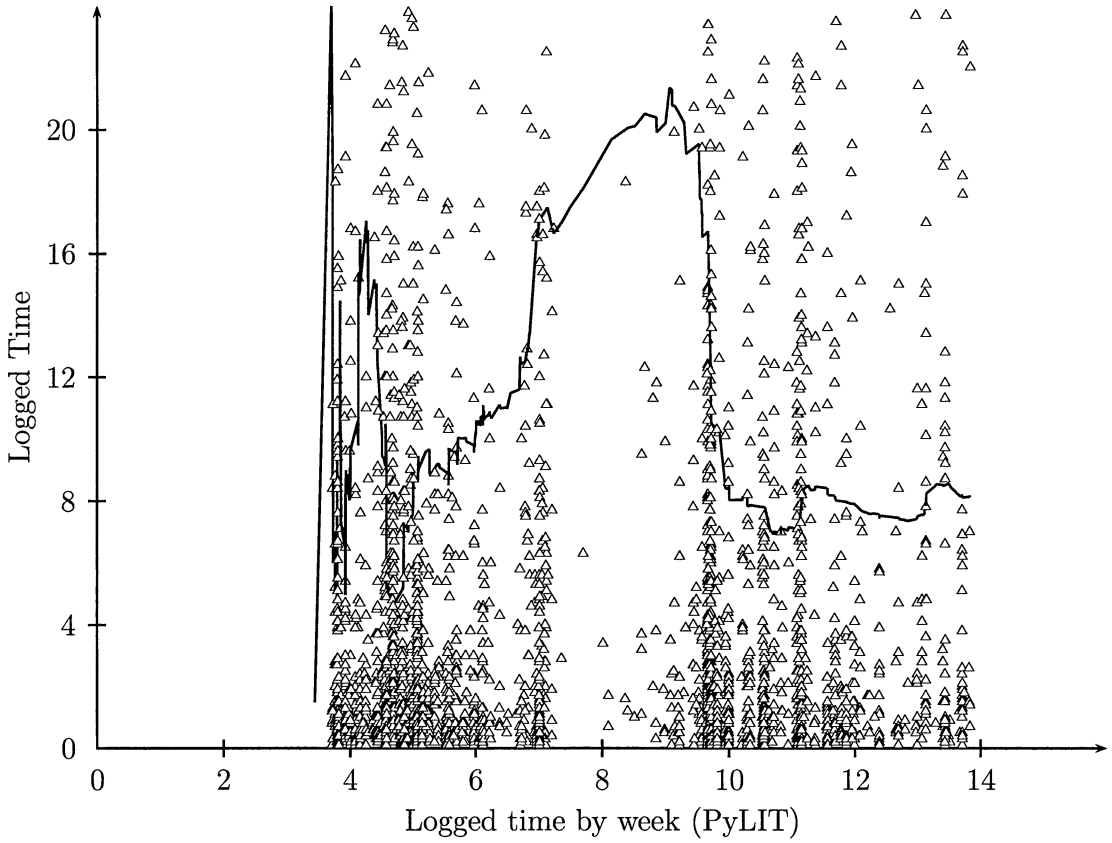

Figure 4.8: 2005 PyLIT Linear Regression Results

In Table 4.12 we see the calculations for the manual collection process. The traditional statistical measures used to assess the quality of the predictions are clearly not very helpful in this analysis. However, we are able to use the linear regression model to predict future times with reasonable accuracy, especially as the study progresses and we collect more data. The same calculations for the data recorded by PSPtimer are shown in Table 4.13. Finally, the data from PyLIT are shown in Table 4.14. As would be expected, the predicted times grow toward the end of the cycle 1 when a project deadline was due. The same results were being seen as the semester ended.

	$\bar{y}$	y_{calc}	a	b	Sr	Sb	Sm	R^2
Week2	67.78	88.16	10.624063.3	-132.88	35.45	3.14	202.07	0.170
Week3	75.03	72.02	-0.580190.3	88.24	77.12	0.99	193.22	0.001
Week4	84.38	107.88	2.494705.3	20.89	77.55	0.55	151.48	0.027
Week5	87.79	120.90	2.262069.3	27.12	80.20	0.46	142.22	0.029
Week6	86.40	113.23	1.295348.3	51.16	80.67	0.43	130.30	0.011
Week7	86.25	107.63	0.793836.3	64.01	79.69	0.36	108.98	0.006
Week8	86.13	95.98	0.316147.3	76.41	77.48	0.27	78.36	0.002
Week9	85.07	85.61	0.014880.3	84.57	75.42	0.23	63.52	0.000
Week10	84.37	80.22	-0.098740.3	87.81	74.56	0.21	57.78	0.000
Week11	90.67	117.92	0.623073.3	66.51	83.98	0.19	52.80	0.022
Week12	90.45	116.62	0.575524.3	67.94	83.81	0.19	52.07	0.019

Table 4.12: Statistical Measures for Manual Data

	$\bar{y}$	y_{calc}	a	b	Sr	Sb	Sm	R^2
Week2	105.38	100.41	-1.928487.3	140.75	104.01	2.65	257.73	0.001
Week3	119.00	156.88	6.203686.3	-16.33	161.27	0.96	326.46	0.013
Week4	115.22	119.64	0.475636.3	103.01	145.23	0.48	218.21	0.000
Week5	114.41	117.04	0.197834.3	108.97	132.90	0.40	208.14	0.000
Week6	118.78	153.45	1.653935.3	72.81	144.08	0.38	205.54	0.006
Week8	118.14	150.65	0.939211.3	91.79	143.96	0.35	186.98	0.002
Week9	117.45	120.71	0.087313.3	114.61	137.50	0.21	101.38	0.000
Week10	120.33	136.17	0.397116.3	106.06	135.14	0.18	87.49	0.003
Week11	118.21	122.02	0.084570.3	115.04	133.51	0.17	80.82	0.000
Week12	118.18	121.86	0.076398.3	115.29	133.11	0.17	79.74	0.000
Week14	118.36	125.16	0.102740.3	114.44	132.73	0.17	77.66	0.000
Week15	117.88	118.75	0.012070.3	117.41	132.51	0.16	74.92	0.000

Table 4.13: Statistical Measures for PSPtimer Data

	$\bar{y}$	y_{calc}	a	b	Sr	Sb	Sm	R^2
Week3	6.72	9.63	2.410888.3	-57.81	14.29	3.32	169.41	0.009
Week4	7.98	8.83	0.211592.3	1.42	17.48	0.36	92.82	0.001
Week5	8.37	10.19	0.202668.3	1.68	19.27	0.22	84.62	0.002
Week6	9.42	17.06	0.528931.3	-8.83	21.44	0.16	72.32	0.017
Week7	9.79	18.09	0.455941.3	-6.46	22.04	0.13	64.69	0.018
Week8	9.77	20.19	0.385817.3	-4.08	21.98	0.12	60.73	0.015
Week9	8.89	8.03	-0.033294.3	10.36	20.66	0.06	29.35	0.001
Week10	8.60	7.15	-0.049655.3	10.95	20.18	0.06	27.00	0.002
Week11	8.75	7.88	-0.026100.3	10.07	20.38	0.05	26.18	0.001
Week12	8.67	7.50	-0.029866.3	10.22	20.17	0.05	25.44	0.001
Week13	8.78	8.14	-0.015106.3	9.60	20.32	0.05	24.54	0.000

Table 4.14: Statistical Measures for PyLIT Data

4.3.3 Predicting Future Item Size

The most important part of the overall PSP/TSP process involves assessing overall progress on a project. Simply examining effort expended in the project is not an adequate measure and management needs to be able to see concrete progress toward a final product.

In any software development project, the final products are code and associated design documents. In the present research, we have not collected data on code sizes. In part this was due to the lack of detailed historical records on file sizes during the course of student activities. Additionally, the PyLIT tool, as currently implemented, is not designed to collect code size data directly, since the code was not being managed by the PyLIT system.

Therefore, the current research focused on the design documentation produced during the course of the project. For the 2004 data, only final report data were available. For 2005, the PyLIT tool was used to produce the data and detailed records were available

There was one issue to address in the 2005 data. That is the fact that the documents produced were in the form of HTML pages. In previous classes, the final documents were written in a word processor, so measures like word and line count could be applied to the

SR	1	2	3	4	5	6	7	8
SRS Cycle1	0	0	2003	2003	2003	2003	2003	3781
SRS Cycle2	7548	4608	4608	4608	4608	4608	4608	4608
SDS Cycle1	0	0	13284	21474	2574	2574	2574	9593
SDS Cycle2	8563	709	709	709	709	709	709	709

Table 4.15: Weekly Document Size Example (Group2)

SRS	1	2	3	4	5	6	7	8
SRS Cycle1	0	0	34545	4296	4296	4296	4296	4296
SRS Cycle2	18983	1878	8374	2685	1869	1869	1869	1869
SDS Cycle1	0	0	1165	710	710	710	710	710
SDS Cycle2	710	67222	67222	1970	1970	1970	1970	1970

Table 4.16: Weekly Document Size Example (Group4)

documents. In an HTML document, the material is tagged, but left in a linear structure that is broken into lines by the viewer's web browser. There was no easy way to produce word or line counts that could be related to the documents from previous years.

In the final analysis, it was decided to break the HTML files up into pseudo-lines consisting of tag pairs. These pseudo-lines could be counted, and a word count produced with conventional tools. The lines are still not what would be displayed in a printed document, but at least we have a usable measure.

Examining the weekly variation on the SRS and SDS documents produced results that followed expectations. For two sample groups, we see activity peaking as project documents came due (at week four and six of each cycle) and then a period of little change in document sizes. Unfortunately, the regression analysis during the first few weeks of each cycle was too noisy to show a trend. However, during the next few weeks, the regression model does provide data that could be effectively used to predict product sizes.

Table 4.15 and 4.16 show these results. from the PyLITdata. Since we had not comparable data for the projects in 2004, we cannot compare these results.

It is apparent that the data collected by PyLIT is of a much more detailed nature than collected by other methods. Perhaps the teams were not managed adequately to encourage development work during the idle periods. The PyLIT system does not currently record details about graphics included in the documents although it does record files uploaded. Much of the activity logged by PyLIT during the times when changes in page size were not occurring may be due to edit sessions where the page editor was activated, but few edits occurred.

As a last examination of individual activity, Table 4.17 shows the editing activity of a typical student working on a single page. This table records the pseudo-line count and page size as the edits occurred. It also shows the number of lines modified (reported as deletes and adds by the tool used to collect these data).

Page editing is a very dynamic activity and not usually examined at this level of detail. It is clear that PyLIT is capable of collecting information about the individuals activities that are much more detailed than that normally used in the PSP/TSP projects. Further research into the exact implications of providing this level of detail are needed to see how to properly integrate the data.

4.3.4 Comparing PSPtimer and PyLIT Activity

The 2005 students were supposed to use the PyLIT system to produce their SRS and SDS documents. Therefore, the activity recorded by both systems was expected to be quite similar in those activity areas covered by both systems. However, that was not the result seen in the data.

A typical comparison of activity for a sample student is shown in Figure 4.9. In this figure, the vertical lines show activity logged by PSPtimer, and the triangles show activity logged by PyLIT. If the student had been using both systems as directed, there would be many instances of overlapping records. In fact we see few such instances, which raises

Page: TopDownDesign						
Date	Added		Deleted		Final	
	Lines	Chars	Lines	Chars	Lines	Chars
Mar 28	0	0	0	0	7	774
Mar 28	1	13	4	25	4	759
Mar 28	0	0	0	0	4	759
Mar 28	3	609	3	621	4	747
Mar 28	0	0	0	0	4	747
Mar 28	128	2592	2	605	130	2860
Mar 28	0	0	0	0	130	2860
Mar 28	60	1280	42	1322	148	2836
Mar 28	0	0	0	0	148	2836
Mar 28	5	81	0	0	153	2922
Mar 28	0	0	0	0	153	2922
Mar 28	3	144	1	36	155	3032
Mar 28	0	0	0	0	155	3032
Mar 28	1	127	155	2877	1	128
Mar 28	0	0	0	0	1	128
Mar 28	8	319	1	127	8	327
Mar 28	0	0	0	0	8	327
Mar 28	1	128	8	319	1	129
Mar 28	0	0	0	0	1	129
Mar 28	8	357	1	128	8	365
Mar 28	0	0	0	0	8	365
Mar 28	1	142	1	141	8	366
Mar 28	0	0	0	0	8	366
Mar 28	1	103	1	142	8	327
Mar 28	0	0	0	0	8	327

Table 4.17: Typical Page Edit Session (Group2)

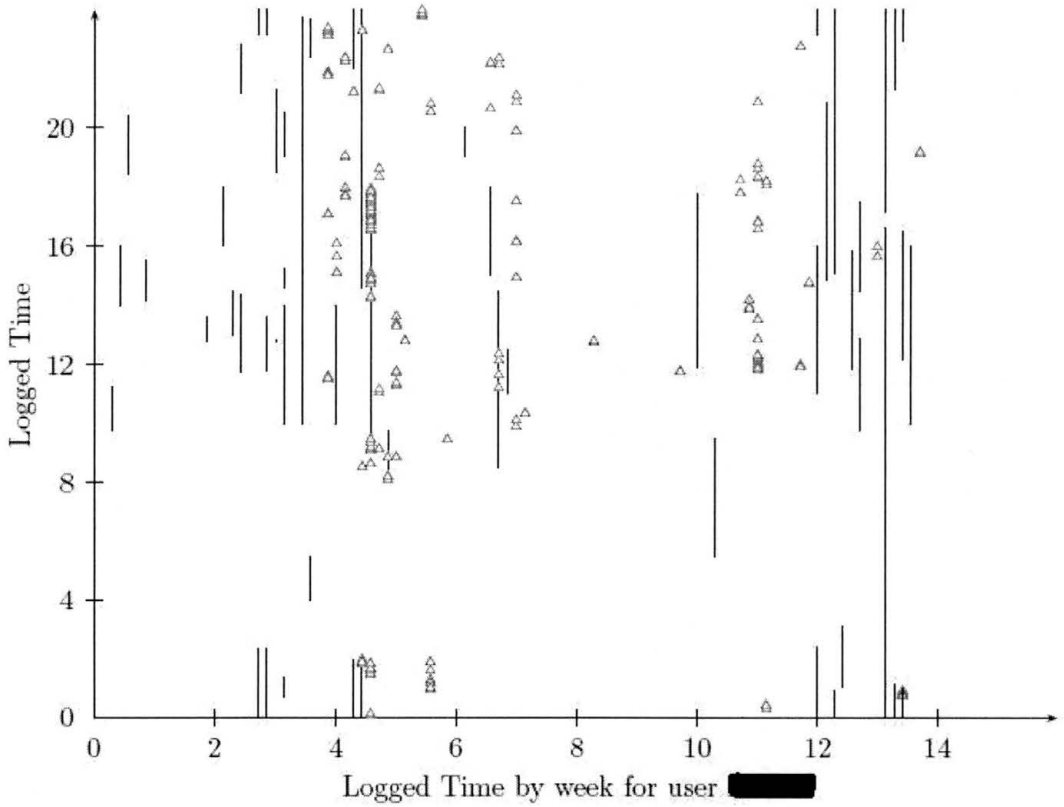

Figure 4.9: PSPtimer and PyLIT Weekly Activity Comparison

questions about the actual student activity. In examining this issue later, it became apparent that the students used the tools they were most familiar with to do some of their writing, and the final results of that activity were added to the PyLIT pages at a later time.

This issue is one that can be addressed with additional training in the use of a system like PyLIT. Many people who start off using wiki systems become addicted to using them as note taking systems, giving u the power of a full document production system in favor of an easy to work with environment. Future work on the PyLIT system will address the ease of use issue.

CHAPTER 5

5.1 Conclusions

This study has shown that using automated timekeeping in a useful collaboration tool can greatly improve the accuracy of the data collected and used in guiding a development project. The study did not attempt to optimize the collection process, but looked at what we could accomplish by adding timing functions to a collaborative tool the engineers would find useful in their normal work. The PyLIT system has proven to be a simple record keeping tool capable of producing good quality time data and project documentation for a software development project.

The tool needs several improvements. For instance, the decision to let the engineers use a JavaScript editor needs to be re-examined. The ease of production of complex HTML documents must be weighed against the increased complexity of extracting useful data from the system. I was not able to find a good HTML processing system capable of delivering statistics on the documents produced by the current PyLIT system. A prior version of PyLIT used more conventional wiki markup, but the students who used that version had mixed reactions to the system. With proper training, the wiki markup can become very natural to use.

We can add to the present system to increase the percentage of project activities that can be covered by the automated data collection process. Code development timing is the biggest data collection problem area in the present PyLIT system. The basic editing capabilities of the Browser, even augmented with a good Javascript editor, are not as productive an environment as developers want to replace good programmer's editors like VI or emacs.

However there is a simple solution. If we move the all or part of the PyLIT system from a central server to the developer's local workstation, and transfer products from the system to a central project server, then we can tap the full power of the toolset available on the engineer's workstation in the data collection process. In this environment, the editing commands could launch the engineer's favorite editor, and track time spent in that system as well. Furthermore, it would be possible to launch command line compilers and collect error messages providing data on defects in the code.

An earlier version of PyLIT, written in PERL, included a simplified web server component enabling the entire system to run locally. The current Python version was designed as a CGI script for the Apache server, but Apache can be installed on a local workstation with little impact on that system's performance. A simple Python web server can also be used to drive PyLIT on a local workstation.

5.2 Future Research

The PyLIT tool alone cannot capture all the data needed to fully use PSP/TSP in a development project. However, it shows great promise in reducing the level of effort needed to collect data when a workstation is available. There are three future areas where additional research would be useful:

- Integrating the PSPtimer functionality with PyLIT
- Capturing Code Development activities
- Adding Support for PDA data capture

5.2.1 Integrating PSPtimer into PyLIT

It would be simple to add the features of the current PSPtimer tool to the basic PyLIT server. With proper data validation, this would allow the combined data from both sources

to be used in a single regression analysis that should give better overall predictive results. This was not done during this study to keep the data from the two approaches to data collection separate. The automatic data logging features of PSPtimer would greatly simplify the record keeping process, potentially eliminating paper logs entirely.

5.2.2 Capturing Code Development Activities

As mentioned above, the largest block of time missed by PyLIT is in code development. The simplistic nature of the editing environment in PyLIT makes it difficult to use for actual code development work. Some early experiments with a variation of Donald Knuth's Literate Programming Knuth (1983) are included in the PyLIT code, and the author has used Literate Programming to develop some of the PyLIT code. With practice, the style of code development encouraged by Literate Programming actually encourages the developer to document the code as it is produced, which can greatly enhance the value of the work produced.

A related experiment designed to capture some code development activity was tested briefly in the current study. The PyLIT tool was extended by adding a component that enabled the engineer to generate graphic images using the DOT program on the server. The developer entered a DOT script describing a figure that would be included in the documentation into s code form displayed on the developer's browser. When that page was saved, PyLIT launched the DOT processor using a Python script. Any error messages produced by DOT were captured and displayed after processing. Once the script could be processed without errors, the image produced was linked into the documentation.

Experimentation with local workstation based versions of PyLIT would clearly be useful to see if the tool can better capture the PSP/TSP data.

5.2.3 Adding Support for PDA Devices

Part of any development project always involves work away from the workstation. Meetings, informal conversations, and research in libraries are examples of times where access to the primary PyLIT system is impractical. However, many developers own Personal Digital Assistants (PDAs) and a simple program on the order of the current PSPtimer application could be used to capture that time data for later synchronization with PyLIT.

APPENDIX A

A.1 Overview of the PyLIT Collaboration Program

In this section, we present a brief overview of the PyLIT application. The complete source to the program is too long to include in this document. It is available from the author.

The PyLIT program is designed as a simple Python CGI script that can be run stand–alone, or under control of any suitable web server. In the development environment, both configurations were tested. A simple stand–alone server can be constructed in a few lines of Python code. Most of the important features of the server are encapsulated in the Python server CGIserver module provided with current Python installations.

The Apache web server was used as a test system for most of the development work and also served as the environment for the production use of the system during this study. This server is available for both the Windows and Linux environments. A Windows installation on the author's laptop served as the primary development system, and code produced there was synchronized with a copy maintained on the Linux server used by the students in this study.

A.1.1 The PyLITCGI Script

The basic structure of the PyLIT program is shown in Figure A.1. As can be seen in this activity chart, page requests arrive at the server from client web browsers and are passed by Apache to the PyLITcgi script. Here data associated with the specific page request are collected and an instance of the Object class is created to hold this information and manage the page request.

Once this Object instance is set up, the CGI script passes the specific page request to an instance of the Handler class. In this class, the page request is processed, and a reply page is formatted. The Handler returns the reply to the client web browser completing the transaction.

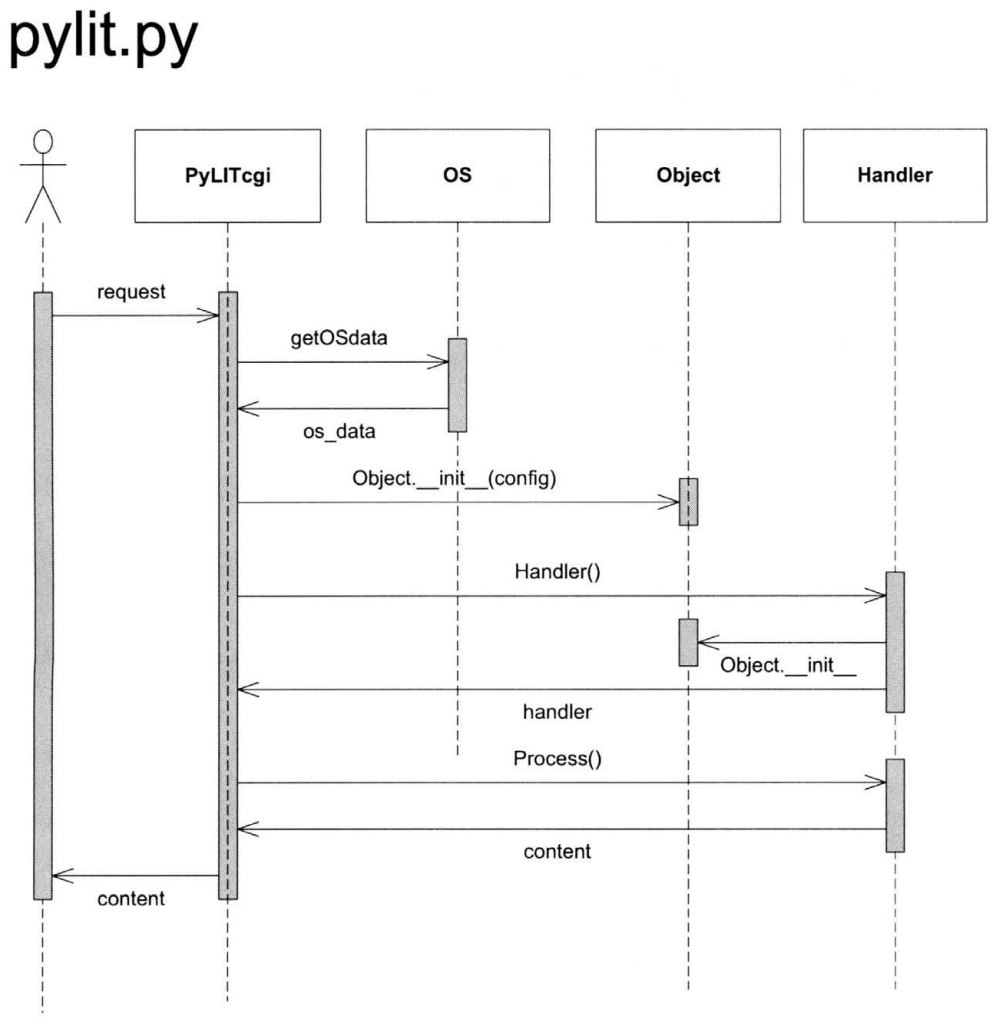

Figure A.1: Main Program Sequence Diagram.

Each request is logged by the Apache server and by PyLIT itself. The basic logging functionality of PyLIT is included in the Object class so all parts of the system can log data as needed. Once a user has logged into the PyLIT system, each request sent to the system

returns a PyLIT generated HTTP cookie containing a session identifier which is used to associate the request with a specific PyLIT user. This information is also logged in the PyLIT database.

A.2 The Request Handler

The Handler class controls all request processing. The activity chart for this class is shown in Figure A.2. PyLIT uses a MySQL database to store the pages created by users and to record all logging entries. The handler creates an instance of the Store class which encapsulates all database processing and manages the database requests. The Handler class also sets up an instance of the Plug-in class (discussed later) to handle specific page requests. All actual processing for any request is passed on to one or more plug-ins designed to perform a variety of actions in response to a single request.

A.3 Page Processing

Once a Handler instance has been initialized the requested page is analyzed to determine if the user has requested a plug-in response, or wishes to view a user defined page. If the request refers to a user page, that page must be fetched from the PyLIT database. If the user has permission to view the requested page, the raw content for the page will be retrieved from the database. If the user request includes a query string referencing plug-ins. then those plug-ins will be activated to post process the page content. When all post processing has completed, the final HTML page content is returned to the user and displayed on the browser.

An overview of this processing is shown in FigureA.3.

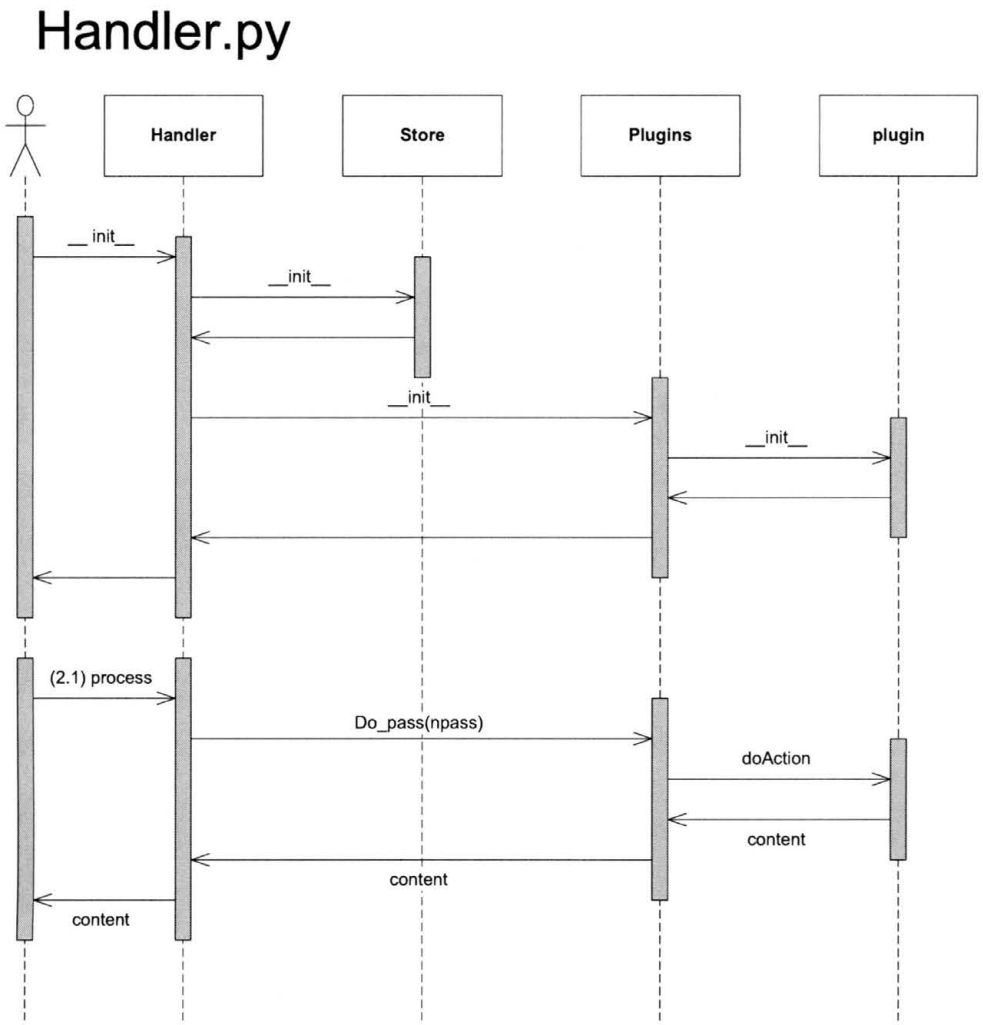

Figure A.2: Request Handler Sequence Diagram.

A.4 Basic Plug-in Logic

The PyLIT request processing system is based on a plug-in architecture which allows the system to be easily extended. The main program does little processing, and is primarily dedicated to establishing a connection to the MySQL database system and loading the currently installed plug-ins. After these actions have been completed, the program enters a loop calling one or more plug-ins to respond to the actual request.

The plug-in loop is designed to provide maximum flexibility by not restricting what

a plug-in does. Some plug-ins are self contained, producing all content needed for a reply, others are involved in formatting content produced by the PyLIT user, or by other plug-ins.

A plug-in is registered in the system during initialization whenever a user page request is received by the server. Plug-in modules are written as Python class files and include a list of specific plug-in identifiers that the module can handle. During the registration process these identifiers are recorded in a simple data structure that ultimately lists all plug-in functions that could be used to process the current user request.

The registration information for each plug-in identifier also includes an integer number used to define the processing loop pass where this plug-in should be activated, and a final boolean value used to exit the plug-in loop if the result of the plug-in processing completely defines the required output.

Rather than fix the number of passes through the plug-in processing loop, as plug-ins are loaded, the pass value is used to set the maximum loop counter. With this scheme, any number of passes can be declared, and the ordering of plug-ins controlled in a simple manner.

Plug-ins may be called by requesting a page whose name is the name of a plug-in, or by requesting any other page and including the name of a plug-in (and other parameters) as part of a query string included with the page request. In this way a plug-in can be called to provide processing for any user page.

When a plug-in finishes working on a page, control returns to the Handler plug-in loop code where the page could undergo still more processing. Once all processing has been completed, the final output is logged, and them merged with a standard template before being returned to the client web browser.

A.5 Login Processing

As an example of how a plug-in works, this section describes the plug-in designed to manage user access to the PyLITsystem. This plug-in receives control when specific pages are requested by the client. Specifically, this plug-in responds to requests for the following page names:

- _loginrequest
- _loginprocess
- _newpwrequest
- _newpwprocess

The sequence diagram for basic login handling is shown in Figure A.4. The plug-in is first activated when the user elects to log in to the system. The request is processed and a login form is returned to the user. Once this form is filled in, the user submits it to the server and the Login plug-in then performs the login validation.

The user name and password are checked in the MySQL database. If the name or password does not match an entry in that database, an error message is attached to the login form and it is returned. If the name and password do match a new session cookie is generated and attached to a welcome page returned for successful login requests. The last step in this process is to update the user information in the MySQL database so future connections from that user can proceed without needing further logins. As presently designed, the session lasts until the user closes the web browser on the client workstation.

A.6 Currently Installed Plug-ins

At present the PyLITsystem includes a number of plug-ins to handle the functions required for this study. Other plug-ins are available to provide specialized functionality for other

projects. A list of the plug-ins used in this study in presented here.

- CodeEditor - Provides for editing code, specifically DOT graphics scripts
- Editor - Provides basic page editing functions (usıng Javascript editor)
- Help - Manages help requests
- LogDisplay - dısplays summary information from the PSPtimer
- Login - Handles all login requests
- PeerManager - Handles Peer Review forms for TSP
- Team - Manages team membership
- Upload - Provides file upload services for graphics files
- User - manages basic user access control issues.

The current page editor is an open-source Javascript system available from Moxiecode Systems (Moxiecode Systems). This editor provides a very familiar editing environment with sufficient power to allow full page formatting including sophisticated table handling. The editor generated conventional HTML pages. However, these pages proved difficult to post process, since a full HTML parser is required to access specific page components. The students found the editor easy to use, until they tried to perform complex cut and paste operations. They also had problems when pasting material generated with other applications into their PyLITproject pages. Overall, it remaıns to be determined if this is the appropriate editing system for future PyLITdevelopment.

A.7 PyLIT Support for PSPtimer

The PyLIT tool was extended by adding a plug-in that displays the time data collected by the PSPtimer application. The time data was displayed in the form of a series of graphs

that could be viewed by individual team members and team managers as needed during the course of the project.

FigureA.5 shows an example of a graph showing how the team time is distributed across the possible PSP development phases.

FigureA.6 Shows the same data distributed according to the development activity.

PyLIT was also extended to allow PSP/TSP Peer Reviews surveys to be conducted. Other standard PSP/TSP forms could be easily added to the system.

Page.py

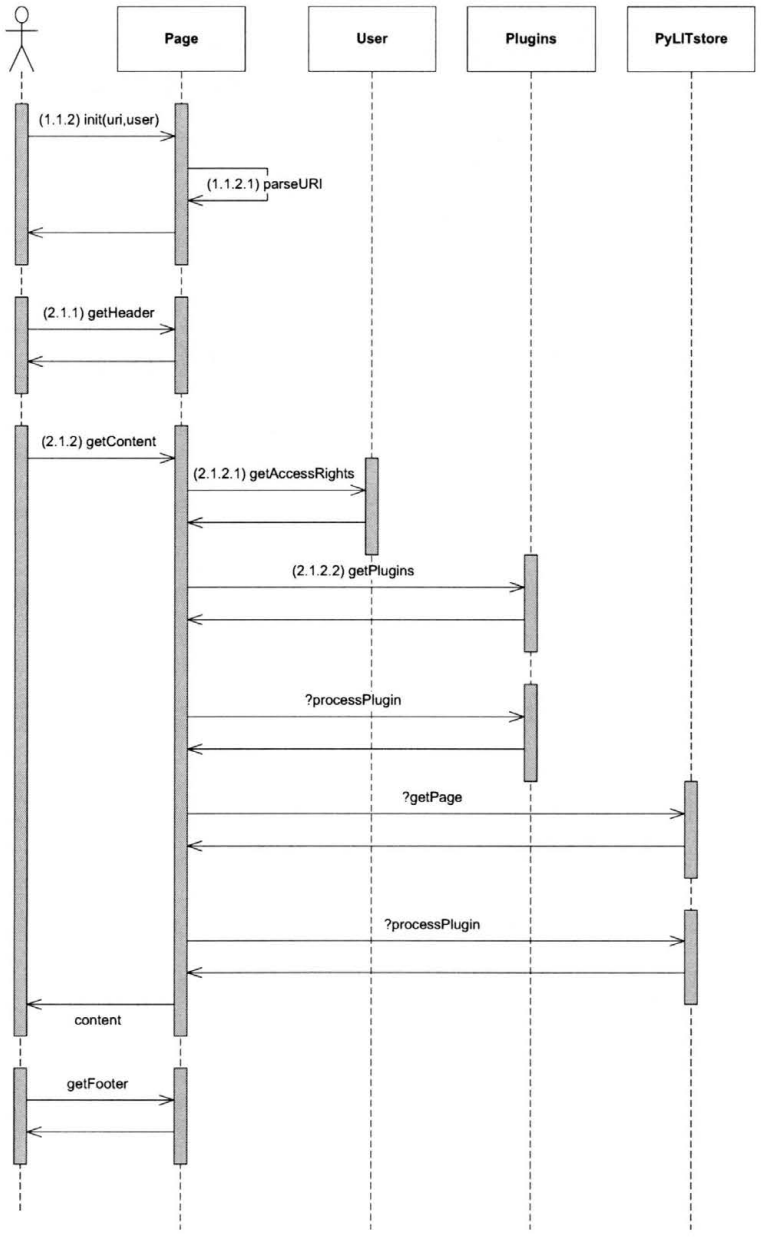

Figure A.3: Page Processing Sequence Diagram.

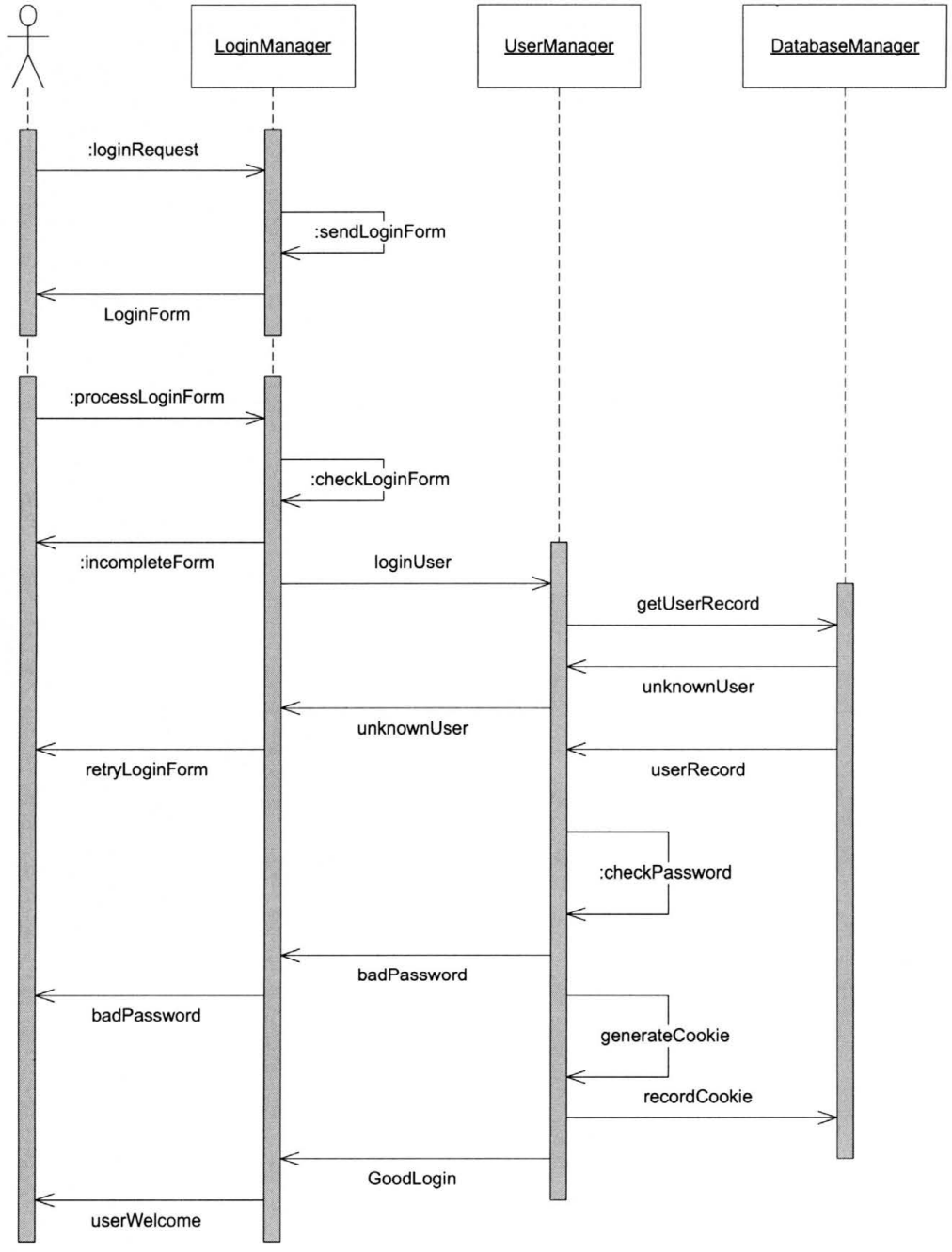

Figure A.4: Login Plug-in Sequence Diagram.

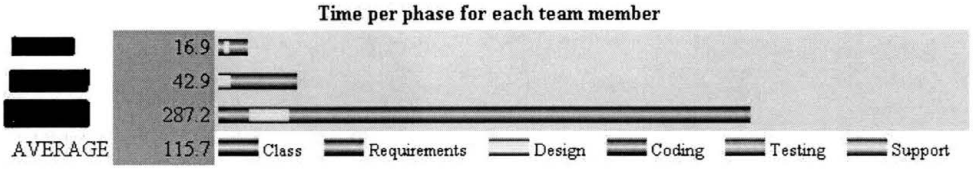

Figure A.5: Team Time by Development Phase.

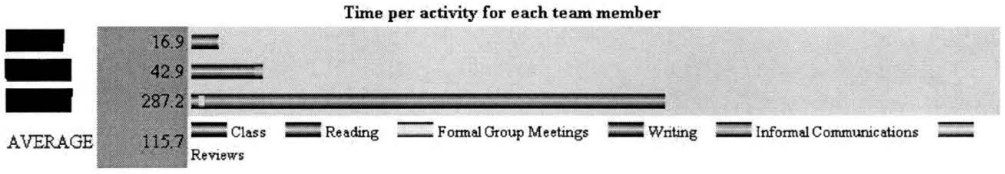

Figure A.6: Team Time by Development Activity.

REFERENCES

R. L. Cannon. Putting the personal software process into practice. In *Twelfth Conference on Software Engineering Education and Training*, pages 34–37. IEEE Computer Society Press, 1999.

N. Davis and J. Mullaney. The team software process(tsp) in practice: A summary of recent results. Technical Report CMU/SEI-2004-TR-014, Software Engineering Institute, Pittsburgh, PA, 2003.

A. M. Disney. Data quality problems in the personal software process. Master's thesis, University of Hawaii, December 1998.

N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley, New York, NY, second edition, 1998.

D. Escala and M. Morisio. A metric suite for a team psp. *IEEE Metrics*, pages 89–92, 1998.

Forbes Corporation. Top 200 fastest computers, May 2004.

G. Hall. An automated tool for psp timekeeping, 2004.

M. Hardy and A. Bryman. *Handbook of Data Analysis*. SAGE Publications, London, England, 2004.

W. Hays and J. W. Over. The personal software process (psp): An empirical study of the impact of psp on individual engineers. Technical report, Software Engineering Institute/ Carnegie Mellon University, Pittsburgh, PA., December 1997.

J. E. Henry. *Personal Software Process Studio*. East Tennessee State University, 1997.

http://processdash.sourceforge.net. Software process dashboard initiative, 2001.

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

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

W. S. Humphrey. Pathways to process maturity: The personal software process and team software process. *SEI Interactive*, June 1999.

iSixSigma LLC. The history of six sigma. 2005.

J. R. Jablonski. *Implementing TQM*. Technical Management Consortium, Inc., Albuquerque, NM., second edition, 1992.

S. Janiszewski and E. George. Integrating psp, tsp, and six sigma. *Software Quality Professional*, 6(4), 2004.

D. E. Knuth. Literate programming. Technical Report STAN-CS-83-981, Stanford University, Department of Computer Science, 1983.

B. Leuf. *Peer to Peer Collaboration and Sharing over the Internet*. Addison-Wesley, Boston, MA., 2002.

Lotus. Overview of lotus notes v6.0. 2005.

H. McCracken. First look: Office xp collaboratıon tools. *PC Magazine*, 2001, May .

M. Morisio. Applyıng the psp in industry. *IEEE Software*, (2000), November/December 2000.

Moxiecode Systems. Tinymce javascript wsisyg editor.

Software Engineering Institute. Tsp and psp frequently asked questions (faq). http://www.sei.cmu.edu/tsp/faq.html, 2005.

VITA

Roie Black was born in 1946, and was raised in the Washington, D.C. area where he developed a passion for technology by spending time in the Smithsonian Institution museums. He attended Virginia Polytechnic Institute and State University, receiving both bachelor's and master's degrees in Aerospace Engineering. Upon leaving graduate school, Roie began a 20 year career as an officer in the United States Air Force. His duties ranged from conducting basic research in computational fluid dynamics to managing operations of a 70 million dollar Cray-2 supercomputer center. After retiring from the USAF, Roie started a computer technology consulting company and served as Director of Information Technology for the Joslyn Art Museum in Omaha, Nebraska.

Roie is currently employed at Austin Community College, as an Associate Professor of Computer Science.

Permanent Address: 11713 Onion Hollow Run

Austin, Texas 78739

This thesis was typeset with LaTeX 2_ε[1] by Roie R. Black, B.S., M.S.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society.