

Tracking Personal Processes in Group Projects

Ly Danielle Sauer,
Sandia National Laboratories
ldsauer@sandia.gov

Timothy E. Lindquist,
Arizona State University
tim@asu.edu

Jeremy Cairney
Arizona State University
cairney@asu.edu

Abstract

Software engineering continues to develop methods for process improvement and quality. The Personal Software Process is one way to introduce software engineers to aspects of process tracking, assessment and improvement. In this paper, we describe the software tools that we've constructed to support the planning and postmortem of software activities. We describe an approach that allows the personal software process to be used in group projects, while still allowing the individual engineer to employ personal process quality and improvement techniques in their own activities. The tools supporting planning and postmortem are used in the context of a workflow system developed at Arizona State University (ASU), called Open Process Components, whose aim is to componentize software services and provide interoperability among various approaches. These tools and approaches explore software development in the increasingly distributed environment in which the software engineer is responsible for their own assessment and improvement.

1. Introduction

Measuring, guiding, and refining an organization's software process improves effectiveness of development resources and provides a level of control for software quality. The development of the SEI Capability Maturity Model [14] has raised awareness of the need for better software processes. Software processes are often discussed at the project management level, and its not uncommon for an organization to employ the services of a process engineer with the intent of wide-scale process improvement.

Software processes describe the interaction among people and artifacts in carrying out the work involved in the software life-cycle. A software process encompasses the work that will be done (activities), what it will use and produce (input and output products), who will do it (agents), as well as, when and how it will be done (behavior). The past decade has seen increased demand for more powerful and robust automated software process

systems. Tool vendors and the research community have responded with a variety of approaches.

A review of the tool market place shows many groupware, process, and workflow tools whose functionality ranges from graphical modeling or simple enactment to full support for defining, executing, analyzing, measuring, and tracking software processes. The Plethora of tools, most of which have not been widely adopted, combines together with the increasingly distributed nature of software development today to form one of the challenges addressed by this paper. That is, the need to have interoperability among a heterogeneous set of process tools (which execute on distributed heterogeneous platforms).

The efforts of the Workflow Management Coalition (WfMC) [16] and the Object Management Group (OMG) are aimed at this challenge. Both organizations are identifying common interfaces that vendors can use for interoperability among their products. Other middleware efforts (Portable Common Interface Set [5],[11] and Open Process Components [8]) are addressing integration of process components with other middleware components (i.e. version, configuration management, etc.). In this paper, we build on these efforts to show how processes can be distributed compositions of personal process components.

Considerable research has addressed automating the software process [1]. Some are addressing formalisms (Petri nets, rule-based, process programming languages, event-based, and object-oriented). Other research includes comprehensive environments centered on processes, such as ISTAR, in which all activities are modeled using a contractual model. In a process-centered environment, nearly all activity takes place within a defined process.

Christie has elaborated several problems in the adoption of process automation [3]. Process-centered environments are typically all-or-nothing and difficult to adopt in steps. Management is justifiably reluctant to invest in dramatic change without a gradual migration path or concrete evidence of value-added. Benefits of enactment support or track the required time consuming front-end resources for process definition. Some systems require definitions of activities that do not have relevance to tracking and improvement. Adoption also places other stresses on an organization ranging from an engineer's

perception of excessive intrusion to the need for additional personnel who specialize in process engineering.

In this paper, we present a process framework that shifts its approach towards composable process components. Project processes are created by brokering among the building blocks of an engineer’s defined personal processes. Software engineers are responsible for tracking measuring and analyzing their own processes distinct from organizational concerns.

2. Overview of Personal Software Process

Current software professionals utilize private techniques and practices that were learned from peers or through personal experiences. Few software engineers are aware of, or consistently use methods that lend themselves to personal process improvement. A personal software development process is a concept introduced to address improvement needs of an individual.

Watts Humphrey of the Software Engineering Institute has formalized a personal software development process called Personal Software Process (PSP) [9],[10]. Today, there are various realizations of PSP to aid software engineers in applying the process. The realizations range from case tools and web-based repository browsers to formal training classes.

2.1. Personal Software Process

PSP is designed to assist software engineers in controlling, managing, and improving their predictability, productivity, and quality. PSP consists of a family of seven personal processes that progressively introduce data and analysis techniques (Figure 1) [10]. Engineers use these data and analysis outcomes to determine their performance and to measure the effectiveness of their methods. Humphrey’s initial result (applied to 50 students and three industrial software organizations) indicates an average test defects improvement of over ten times and productivity improvements of better than 25% [9].

Figure 1 shows the PSP progression in which each PSP step includes all the elements of prior steps together with additions. The PSP process steps are Baseline Personal Process (PSP0, PSP0.1), Personal Planning Process (PSP1, PSP1.1), Personal Quality Management (PSP2, PSP2.1), and Cyclic Personal Process (PSP3). Starting in *The Baseline Personal Process*, the software engineer creates the foundations for measurement and improvement. PSP0 is the software engineers current software development process extended to provide measurements (time and defect trackings). PSP0 covers three phases: planning, development (design, code, compile, and test), and postmortem. PSP0.1 includes

coding standards, size measurements, and a Process Improvement Proposal (PIP).

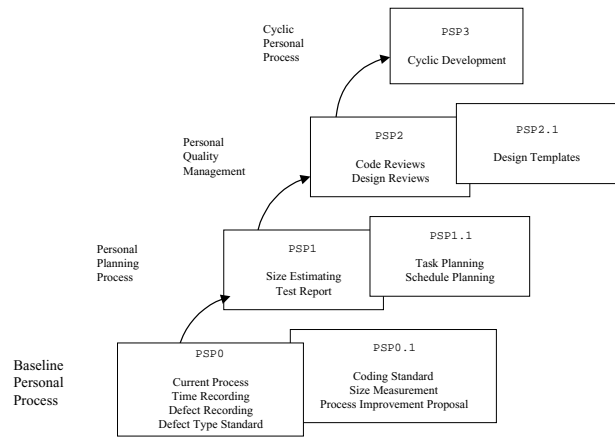


Figure 1. PSP Process Evolution

Personal Planning Process (PSP1, PSP1.1) adds planning to the baseline. Here, the software engineer prepares the basis for project tracking, which include software estimates and development plans. The goal is to learn the relationship between program size and resources, as well as how to make realistic schedules. PSP1 enhances PSP0 and PSP0.1 to include size and resource estimation and a test report using Proxy Based Estimation (PROBE) as a method to estimate sizes and development times.

Personal Quality Management (PSP2, PSP2.1) provides defect management by tracking the relationship between time spent in reviews and the phases during which defects are injected and removed. Prior project defect data are used to realize review checklists and self-assessments. PSP2.1 extends PSP2 with design specifications and accompanying analyses. The goal is to provide the criteria for design completion.

Cycle Personal Process (PSP3) introduces techniques for developing large-scale projects. The approach calls for sub-dividing into personal processes. Development is done in incremental steps starting with a base module.

2.2. Personal Software Process Studio

Personal Software Process Studio (PSP Studio or PSPS) [6] is a case tool developed at East Tennessee State University to assist in using the Personal Software Process. PSPS does so by automating the planning and postmortem artifacts. In particular, PSPS provides the following features: Data Measurement, Historical Database, Convenient Access to Tables, Statistical Calculations, and Guidance through the Process. The *Data Measurement* feature allows developers to accurately (similar to a stopwatch) measure development times, track defects, and measure program sizes. The *Historical*

Database feature allows developers to store all of the developers historical PSP data in a reliable and secure database. *Convenient Access to Tables* provides a window with tab access to the forms. The *Statistical Calculations* feature automatically maintains totals and performs the statistical calculations. The *Guidance through the Process* feature provides on-line directions for using the PSP.

PSP Studio groups all of the automated paper work, forms, and calculations into two categories: Process Tables and Project Tables. Process Tables guide or improve the individual software engineer process with an online process outline, access to standard tables for defects, LOC and coding standards, and access to a process improvement proposal.

Project Tables are automated forms, such as *Logs*, *Summaries*, and *Templates*. The log tables support tracking time, defects, and issues. The Project Summary table records the estimated and actual totals for the project and for all projects to date. The Cycle Summary table supports the project summaries by capturing the planned and actual size, time, and defects for each cycle.

2.3. ECEN 4553 Database Browser

ECEN 4553 PSP Database (PSP Database Browser) [2] is a web-based database that also automates many of the PSP forms, scripts, calculations, and reports. The database is organized to capture a set of related data for an individual software engineer. The core of the database is the concept of a job, which is a software engineer's activity. Once the job is defined, the software engineer can log time against the job, log defects against the job, and specify a detailed project plan for the job.

Although the PSP Database Browser does not strictly adhere to all of Watts Humphrey's Personal Software Process data, it collects planned and actual data for each job. ECEN 4553 PSP Database Browser automates a subset of Watts Humphrey's Personal Software Process with a web-based user interface.

3. Applying PSP to Group Projects

ISO 9000 [4] and the Capability Maturity Model (CMM) [14] assist organizations in improving their processes. Personal Software Process, on the other hand, provides an improvement technique for software engineers in the context of individually developed software. Seamless integration of the PSP within a software organization cannot be achieved, since individuals rarely cycle through all phases of development on a software project. Engineers can, however, apply PSP analysis techniques to their individual activities on group projects. The resulting metrics can be the basis for personal process improvement, without having the "big

brother is watching over me" complex that is common to organizationally imposed quality and improvement efforts. This section discusses our approach to providing well integrated organizational and personal process improvement.

At ASU, we have been developing software to support the use of planning and postmortem phases of the PSP and to support their application to various life-cycle activities. For example, in an organizational setting, an individual may be assigned to testing. The test engineer would develop their own test process that includes planning and postmortem. The resulting personal test process becomes part of an organizational or project process. The "integratable" personal processes (personal test process) collect product measures, use defect analysis, and consider resource usage as a means of improving that process segment. The artifacts and the automation we have developed are discussed in Section 3.2.

3.1. Process Components

The Open Process Component Toolset (OPC) [8],[12],[13] is a set of tools developed at ASU to support process definitions and enactment. OPC's basic premise is that a process is a process component and may consist of one or more process components. Process components may be compositions of subcomponents whose underlying representations may differ. For example, a Process Weaver component, called *create_design*, may be composed with a TeamWare Flow component called *review_design*. Thus, the Integrated Process is defined as a process component consisting of three process components: the Planning Process Component, the Personal Software Activities Component, and the Postmortem Component. The Software Activities component may be any process component such as, testing, design, coding, or review.

Discussion of these components and the support we have implemented for Planning and Postmortem artifacts can be found in Sections 3.1.1, 3.1.2, and 3.1.3).

3.1.1. The Planning Process. The *Planning Process Component* defines the individual engineer's plans for the software activity. The process is assigned to the project planner, takes as inputs the customer requirements (written or oral) and produces as output an initial version of the planning artifacts, a requirements specification, a cost estimate report, and a size estimate report. Additionally, an engineering notebook for the project is created and initialized based on the activity schedule. At this phase, the project activity schedule and the project plan summary forms only contain planning information such as estimated total size, the project development duration, and defects injected and removed. Further

descriptions of the project activity schedule, the project plan summary, and the engineering notebook are discussed in Section 3.2.

The Planning Process Component is composed of its children process components: Identify Requirement, Perform Size Estimation, Perform Cost Estimation, and Construct Plan. Each child process component is defined to perform a specific task to help planning the software activity and laying the groundwork for analysis. For instance, the *Identify Requirement Process Component* generates the requirement specification (SRS) given the customer requirements.

The OPC definition tool uses graphical depiction to model the Planning Process [12]. The model includes nodes for *Processes* (process components that have sub-components), *Activities* (process components without subs), *Roles* and *Products*. Directed edges depict relationships (i.e. *is_input_to*, *has_output* and *has_sub*). For example, the Requirement Specification (*Product*) *is_input_to* Perform Cost Estimation (*Activity*), and the Identify Requirement (*Activity*) *has_output* which is the Requirement Specification.

3.1.2. Personal Software Activities. In the PSP, the planning and postmortem activities depend on a personal software process that includes the phases: planning, design, code, code review, compile, test, and postmortem. In our application of the PSP to group projects, we provide the capability to replace design, code, code review, compile, and test with other activities. Our approach is to provide the background for the planning and postmortem phases as applied to any software related activities. In a group project, an individual engineer may not be involved in coding, compiling, and testing, but may instead work on design and design reviews, or may instead be a test engineer whose involvement does not go beyond planning, developing, executing and reporting on tests. Our assumption is that the analysis techniques that consider resources (labor, primarily), product measures and quality assessment all equally apply to any other software related activities, whether directly developing code or not. Process improvement should be a center of focus for all participants in a software process.

At ASU, we have been using this approach to Integrating Personal Processes for group software projects in a classroom setting and for group independent study projects. Thus far, uses are for small applications in which most project members get involved with all of the life-cycle activities.

The primary challenge in generalizing the approach to large group efforts has to do with product and quality measures. PSP relies on Source Lines of Code as the basis for product measures. Software defect management is the basis for quality, planning, and process improvement. Engineers using PSP, record defects by type, phase

injected and phase removed. PSP uses yield (percentage of defects removed before compiling), appraisal cost of quality and failure cost of quality as the primary input for quality management and process improvement. These are good starting points for the practicing software engineer, however, one must define product measures and defects in a manner appropriate to the activity. For example, a test engineer may use test cases generated as the primary product measure. For example, test cases may be defined to be triples (input condition, action, expected result) independent of how the test case is realized in performing tests. Defect types for a test engineer may include: unsatisfied test requirement and resulting software defects for which there existed a test case.

3.1.3. The Postmortem Process. The *Postmortem Process Component* defines a process for analyzing the performance (postmortem analysis) of a completed project. Postmortem analysis gathers product measures, performs actual resource usage analysis, performs actual defect analysis, and performs summary quality analysis.

We have used OPC to depict Postmortem. The process is assigned to the process engineer and accomplishes its objective of producing the project plan summary by using the initialized project activity schedule, the project defect log, and the initialized project plan summary as input products. Unlike the Planning Process Component, Postmortem does not use children process components to accomplish its goal.

3.2. Automated Support for Process Artifacts

The Integrated Personal Process uses four artifacts: the Engineering Notebook, the Project Activity Schedule & Log, the Project Defect Log, and the Project Plan Summary. We have implemented each artifact as a stand-alone application. When using the worklist handler of OPC, enacting one of the Integrated Personal Process Planning or Postmortem activities may cause the invocation of one or all of these applications according to the process input and output specifications.

All four artifacts use a single repository interface to store and manipulate data. The interface is implemented in Java, using synchronization to support multiple concurrent accesses. Highlights of these artifacts are detailed in the following sections.

3.2.1. The Engineering Notebook. The Engineering Notebook is an application, which implements some concept of the Personal Software Process Engineering Notebook. The Engineering Notebook objective is to create an engineering notebook that tracks the software engineers daily time usage. More specifically, as shown in Figure 3, the Engineering Notebook allows a software

engineer to define and record, for a given project, its activities, the time spent on the activities, and product lists of the activities.

An activity is a unit of work which takes an engineers time (e.g. interruptions, coding, breaks, lunch, designing, etc.); it is any work performed by a software engineer. The time spent on each activity is recorded in an increment of hours; for instance, a job that takes 15 minutes could be recorded as 0.25 hours, but our usage generally limits granularity to one tenth of an hour (6 minutes). The product list entry allows the engineer to list products produced by the activity. The initial engineering notebook is derived from information in the Project Activity Schedule & Log (Section 3.2.2).

Activity	4/5	4/6	4/7	4/8	4/9	4/10	4/11	Product	Week	Project
Learn	0.8	1.5	2.8	0.0	0.0	0.0	0.0		3.5	6.0
Learn	1.8	1.0	0.8	0.0	0.0	0.0	0.0		2.0	4.0
Design	0.8	1.0	0.8	2.0	0.0	0.0	0.0		3.0	11.0
Read	0.8	0.0	0.8	0.0	0.0	0.0	0.0		3.0	0.0
Planning	0.8	0.0	0.8	0.0	0.0	0.0	0.0		3.0	0.0
Design	1.8	0.0	0.8	0.0	0.0	0.0	0.0	Design	1.0	8.5
Design	0.8	2.5	0.8	1.0	0.0	0.0	0.0		3.5	6.0
Coding	0.8	0.0	0.8	0.0	0.0	0.0	0.0		3.0	0.0
Code	0.8	0.0	0.8	0.0	0.0	0.0	0.0		3.0	0.0
Complete	0.8	0.0	0.8	0.0	0.0	0.0	0.0		3.0	0.0

Figure 2. Engineering Notebook Main Window

Modification to the times in the engineering notebook causes the transfer of the times spent and the product list to the Project Activity Schedule & Log.

3.2.2. Activity Schedule & Log. The Activity Schedule & Log (ASL) is an application, which aids in developing project plans. The ASL application allows the project planner to identify the activities, phases, agents, and times for the activity. When the user completes the activity specifications, ASL places the schedule in a persistent repository.

An activity is a task of the project. A project may have a set of activities (process components) representing the work of all group members assigned activities on the project. Similarly, all engineers working on a project will have their own activity schedules, which reflect the lower-level activities necessary to complete their input to the group. The lower-level activities are subject to analysis and improvements as defined above.

A project planner may also have an ASL to coordinate the activities and products of a group of engineers. The Process Broker [15] can be used to determine the kind of activities that the project may need and to check for the availability of those process components. To do this, the project planner first specifies the characteristics of the current project to the Process Broker. The Process Broker uses its locating and matching semantic engine and its

repository of process components to determine the projects that best fit the specified criteria.

For each activity, the project planner estimates a begin and end time, the development duration, the project size, and the output products. These are estimated values, thus, the project planner can use experience to determine the values, some tools, or historical project data. OPC is designed to allow add-on functions. For example, the user can add the tools for estimation or a LOC Counter. The add-on tools are specified using MIME types.

3.2.3. Defect Log. The Defect Log (DL) is an application, which automates Defect Recording to aid in tracking defects injected and removed. The defect data are stored in the defect log, which are used as input to generating the plan summary (Section 3.2.4) in the postmortem analysis. The Defect Log is realized as a tabular application where the rows represent the defects and their information and the columns are classifications of the defects. As shown in Figure 4, the DL allows its user to specify the date, the defect type, the injected phase, the removed phase, the fixed time, and a description.

Date	Type	Inject	Removed	Fix Time	Description
3/10/1998	Logic	Planning	Design		2 Bad design
4/10/1998	Syntax	Coding	Review		3 Missed lots o...
4/10/1998	Syntax	Coding	Review		3 Missed lots o...
4/10/1998	Syntax	Coding	Review		3 Missed lots o...
4/10/1998	Syntax	Coding	Review		3 Missed lots o...

Figure 3. Project Defect Log Main Window

The date that the defect was discovered and the defect description can be anything that the user enters. The DL default defect types are: Documentation, Syntax, Build/Package, Assignment, Interface, Checking, Data, Function, System, and Environment. These can be modified to allow application of planning and postmortem to any software activity. Additionally, the DL also provides default phases including: Planning, Design, Code, Review, Complete, Test, and Postmortem. Analogous to the defect types, these can be changed to accommodate the activity. Finally, the time required to correct the defect is recorded in hours and tenths.

3.2.4. Project Plan Summary. The Plan Summary (PS) is an application, to aid in planning and tracking a software activity. The plan summary is initialized in the planning activity and is completed in postmortem. In our implementation, information in the plan summary is derived from the ASL. The plan summary can be saved

and named so that an engineer who participates in several software activities (reviews, testing, and coding, for example) can track data specific to the activity.

4. Current and Future Work

OPC provides an initial set of tools for defining and enacting process components. The underlying implementation of OPC provides the framework for wrapping various process tools for interoperability. We have achieved initial wrappings of two products, and hope to soon demonstrate interoperability between these products in the near future. Thus, a process component can be defined in terms of sub components each under the direction of a different enactment engine.

We have used our integrated personal software process approach in classroom projects and in group independent studies. The tools described in this paper will be introduced to these projects. Enactment using OPC is controlled by a worklist handler tool, which connects to a repository of process components. Process components are all represented using Java objects. Until the tool wrappers are fully functional, enactment involves launching an application associated with the input and output products as specified with MIME types.

A few important distinctions differentiate our approach to integrating personal processes. Engineers are not asked to carry out a defined process that they themselves have not developed. Engineers are motivated to use process improvement techniques, since they directly and solely apply to their own activities. Product and defect measures are defined by the engineer and thus problems of consistency do not arise. Engineer define their own personal process for the software activities they perform. These may defined or applied from definitions they obtain from other engineers.

5. References

- [1] P. Armenise, et al., "A Survey and Assessment of Software Process Representation Formalisms", Intl. Journal of Software Engineering and Knowledge Engineering, vol. 3, no. 3, pp. 401-426. 1993.
- [2] L. Carter, ECEN 4553 PSP Database Browser, University of Colorado at Boulder, ece-www.colorado.edu/~ecen4553/Reference/psp/examples.html.
- [3] A. Christie, et al. "A Study into the Current Usage of Software Process Automation", in Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems, Athens, GA, May 1996.
- [4] F. Coallier, "How ISO 9001 Fits into the Software World", (IEEE Software, January 1994, pp. 98-100).
- [5] J.C. Derniame, et al. "Life-Cycle Process Support in PCIS, Or It Is Time to Think about Software Process Formalisms Standardization", in Proc. of the PCTE'94 Conf. PCTE Technical Journal No.2, PIMB Assn, November 1994.
- [6] East Tennessee State University, *Personal Software Process Studio*, (East Tennessee State University, www.cs.etsu.edu/softeng/psp/dlpsps.html).
- [7] K. Gary, "Process Interoperability with Open Process Components", Arizona State University, Ph.D. Dissertation, January 1999.
- [8] K. Gary, T. Lindquist, L. Sauer, and H. Koehnemann, "Automated Process Support for Organizational and Personal Processes", in Proc. of the Intl. Conf. on Supporting Group Work, Phoenix, AZ, 16-19 Nov 1997.
- [9] W.S. Humphrey, *Introduction to the Personal Software Process* (Reading, MA: Addison-Wesley, 1997).
- [10] W.S. Humphrey, *A Discipline for Software Engineering* (Reading, MA: Addison-Wesley, 1995).
- [11] The US-France Technology Research and Development Project, *PCIS2 Architecture Specification Version 1.0*, (Lindquist, TE editor) SPAWAR Systems Command, San Diego CA, January 1998.
- [12] T. Lindquist, "A Toolset Supporting Distributed Process Components", Arizona State University, Technical Report, TR-97-034, 1997.
- [13] T. Lindquist, and J.C. Derniame, "Towards Distributed and Composable Process Components", in Proc. of the European Workshop on Software Process Technology, June 1997.
- [14] M.C. Paulk, B. Curtis, and M.B. Chrissis, "Capability Maturity Model, Version 1.1", (IEEE Software, July 1993, pp. 18-27).
- [15] L. Sauer, "Brokering of Process Components", Arizona State University, Ph.D. Dissertation, February 1999.
- [16] The Workflow Management Coalition. The Reference Model. WfMC Document Number TC00-1003, January 1995.