

Personal Software Process in the Database Course

William I. Bullers, Jr.

Anderson Schools of Management
University of New Mexico
Albuquerque, NM, U.S.A.

bullers@unm.edu

Abstract

This paper describes the integration of the Personal Software Process (PSP) into an introductory database course in an MIS curriculum. PSP is a highly disciplined, process-based approach that serves to guide an individual programmer's activities during the development of a software system. Developed at the Software Engineering Institute (SEI) at Carnegie Mellon University, PSP defines how to use process principles to plan, track, and analyze personal programming work; measure effort, software product size and defects; manage software quality; and improve personal programming practices. Although generally used to augment programming courses in computer science, the PSP is more generally applicable to software development in a variety of contexts. This paper discusses an introductory database course for which PSP was adapted and integrated in the database development process. Student and instructor perceptions concerning the revised course are discussed.

Keywords: Database Course, Personal Software Process, PSP, Pedagogy.

1 Introduction

The Personal Software Process (PSP) is "a structured framework of forms, guidelines, and procedures for developing software" (Humphrey 95). It represents a highly disciplined, process-based approach to guide an individual programmer's activities during the development of a software system. PSP defines how to use process principles to plan, track, and analyze personal programming work; measure effort, software product size, and defects; manage program quality; and improve personal programming practices.

The objectives of PSP are to introduce software engineers to a process-based approach to developing software; to show engineers how to measure, estimate, schedule, and track their work; how to improve the quality of their programs; and how to use process and product data on effort, size, and defects to improve their personal programming process (Humphrey 1997). Although generally used to augment a one or two semester programming course sequence, the PSP is more generally

applicable to software development in a variety of contexts. Its basic principles extend beyond programming tasks to include systems analysis, design, and database development activities.

The purpose of this paper is to discuss the integration of PSP practices in the introductory database course in an MIS curriculum. First, an overview of the PSP and its organizational context is presented. Next, the recommended approach to the introduction of PSP in the CS and MIS curricula is discussed. This is followed by an overview of the basic content and structure of the database course. The integration and adaptations necessary for the PSP to be covered in the database course are then presented. Finally, perceptions on the use of PSP for database development are discussed, encompassing perspectives of both students and the instructor.

2 PSP in Organizations

2.1 Organizational Capability Maturity

The PSP owes its contextual perspective to the Software Engineering Institute's (SEI's) Capability Maturity Model for Software (SW-CMM) (Paulk, Weber, Curtis & Chrisis 1995) and Capability Maturity Model Integration for Software Engineering (CMMI-SW) (CMMI-SW 2002). The CMM's provide a framework for assessing the maturity of an organization's software development processes and for identifying a set of "best practices" for software development. The original CMM focused strictly on software development. The successor CMMI project generalized the CMM framework for software into disciplines such as systems engineering, integrated product and process development, and supplier sourcing in order to provide general process improvement guidance for organization.

CMM's can be used for two basic purposes. An organization can use a CMM as a "scorecard" to assess how its software development practices compare to other organizations (Hotle 2003). Through an appraisal process an organization can identify its strengths and weaknesses, or perhaps determine its CMM rating to use as a marketing tool in a competitive bidding situation. Alternatively, an organization can use a CMM as a "road map" to provide guidance for a software process improvement (SPI) program.

The CMMI-SW consists of 22 process areas (PA's) grouped into 5 levels of progressive maturity that characterize the organization's software development capabilities. The "initial" or lowest level of maturity is

simply one of ad hoc software development in which no PA's are institutionalized. The 2nd, or "managed" level, focuses on establishing repeatable project performance in the software organization. This can be accomplished by establishing organizational processes in 5 key PA's encompassing requirements management; project planning; project monitoring and control; supplier agreement management; product and process quality assurance; and configuration management, measurement, and analysis. Similar PA's are defined for the more mature "defined", "quantitatively managed", and "optimizing" levels of the CMM-SW. The levels and focus of each are summarized in Table 1.

LEVEL	FOCUS
Optimizing	Rapidly Reconfigurable Organization Performance Qualitative, Continuous Process Improvement
Quantitatively Managed	Improving Organizational Performance
Defined	Improving Project Performance
Managed	Repeatable Project Performance
Initial	Ad hoc

Table 1: CMM Levels and Focus

The 5 levels and 22 PA's are augmented by a set of goals and practices that can be grouped according to common features. There are 4 common features in a CMMI model: commitment to perform, ability to perform, directing implementation, and verifying implementation. A sample generic practice specified for the "directing implementation" common practice is to "collect improvement information". A number of subpractices are also described in the CMMI-SW, serving to amplify each practice. The subpractices defined for "collect improvement information" include "store process and product measures in the organization's measurement repository", "submit documentation for inclusion in the organization's process asset library", "document lessons learned from the process for inclusion in the organization's process asset library", and "propose improvements to the organizational process assets".

Specific goals are also prescribed, each applying to a PA and addressing unique characteristics that describe what must be implemented to achieve the PA. A specific goal for the "requirements development" PA in the "defined" level is to "develop customer requirements". This goal can be achieved through specific practices, including "elicit stakeholder needs, expectations, constraints, and interfaces for all phases of the product life cycle" and "transform stakeholder needs, expectations, constraints, and interfaces into customer requirements". Goal subpractices are also described that support the achievement of a specific goal practice.

The CMMI provides two distinct paths for improving process maturity. The CMMI defines a staged representation in which the PA's are grouped into levels.

This representation is similar to the original SW-CMM, indicating which PA's must be implemented to achieve each maturity level. Thus the maturity levels describe a process improvement path that illustrates the SPI evolution for the organization. The CMMI also provides an alternate path to process (or product) improvement through a continuous representation. Through this path, the advancing levels of capability maturity are defined within each PA in each organizational unit.

Thus, the 5 progressive levels defined in Table 1 provide a map of an organization's path to advanced maturity. An organization can institute repeatable practices in order to advance from the initial to the managed level; adopt common development processes to progress to the defined level; utilize quantitative measures for process control to move to the quantitatively managed level; and finally institutionalize continuously improving practices to advance to the optimizing level.

2.2 The Role of PSP in the Organization

The CMM's describe "what" an organization must do in order to upgrade its software development capabilities, but they do not address "how" the organization should go about doing so. In order to have a high-performance software organization, the organization must have high-performance software engineers working on high-performance software teams. The SEI developed the Personal Software Process (PSP) (Humphrey 1995) and the Team Software Process (TSP) (Humphrey 2000) in order to provide specific guidance on how individuals and teams can use CMM process principles in their work.

The PSP provides a set of operational process scripts to partition programming activities and responsibilities; describe how to collect and analyze programming process and program product data; and define how to measure program size, defects, and effort involved in designing and coding software programs. The PSP covers a number of planning topics, including how to estimate and measure program sizes, estimate resource usage, and measure process fidelity. These planning topics are supplemented by quality topics associated with the program design process and design verification, including defect identification, measurement, and management.

PSP follows an evolutionary approach in which the program development process and basic measures are first introduced, then supplemented with coding standards and process improvement guidelines. Next, prescribed methods for size estimating and test reporting are added, including planning methods for describing software development tasks and preparing individual work schedules. Code and design reviews are then added to the development process to help manage defects. Finally, cyclic development methods are imposed on the software development process.

SEI teaches the full-fledged PSP (Humphrey 1995) to professional software engineers in a rigorous, two-week training program that costs approximately \$10,000 per engineer. A number of academic institutions also teach PSP in a semester-long software engineering course. Students learning the full PSP complete a series of 10

increasingly complex programming assignments and write 5 reports, applying increasingly sophisticated PSP techniques to the program development process. Staged introduction of prescribed PSP techniques throughout the 10-program series and cyclic development of programs are important principles inherent in the PSP approach to the programming process. Engineers first gather data to establish baseline measures describing their current programming practices. They then learn personal planning practices and quality management techniques. Finally they refine those practices through repetitive, cyclic development practice. The repetitive recording, tracking, and analysis of process metrics serve to reinforce the discipline required in adopting the PSP methods.

The PSP requires that software engineers track, record, and analyze voluminous data concerning their personal programming practices. Many engineers regard the course-work to be difficult and time consuming, and the data tracking process to be quite onerous (Starret 2000). However, the SEI has been able to document a number of benefits attributable to PSP based upon such data (Hayes & Over 1997; Ferguson, Humphrey, Khajeroori, Macke, & Matvya 1997). PSP-trained software engineers have been shown to have higher program quality, reduced development cycle times, improved software processes, costs and schedules that more closely track plans, and improved team productivity. In addition, the SEI core set of inter-related metrics encompassing functionality (usually expressed as measures of size), schedule time, effort (convertible to cost), and defect rates provide a basis for evaluating productivity between projects and over time (Putnam & Meyers 2000).

The TSP extends the CMM and PSP methods to define process principles for building self-directed teams to establish software development goals, plan and track their project work, and accelerate SPI. The TSP defines team member roles to be undertaken during a software development project, prescribes coaching and motivation for team members, and provides a series of scripts that describe each phase in the software development life cycle. The scripts also provide entry and exit criteria which must be satisfied for each phase in the development process. While the TSP has not been in usage as long as the PSP, early adopters have reported positive results (Webb & Humphrey 1999).

3 PSP in the CS and MIS Curricula

Many CS programs begin with a two-semester introduction to computer programming, often referred to as CS1 and CS2 (Computer Programming I & II), as originally outlined in *Curriculum '78* (1979). PSP can be taught as a separate course in software engineering practices following CS2 (Borstler, Carrington, Hislop, Lisack, Olson, & Williams 2002). The approach in this case is to use the same PSP text (Humphrey 1995) that SEI uses in its industrial training classes. However, few academic institutions have the resources to devote a separate course to PSP.

More often, an introductory version of PSP (Humphrey 1997) is appended to CS1 and CS2 (Grove 1998; Hou & Tomayko 1998). PSP time management and planning is integrated with the techniques of algorithm development and programming in CS1. PSP quality management is then appended to the continuing development of program design, debugging, and testing in CS2. In this approach, students apply a subset of the PSP techniques to their regular programming assignments in CS1 and CS2.

The expanding scope of computing, detailed in *Computing Curricula 2001 Computer Science* (CC2001-CS) (2001), has led some academic programs to implement a three-semester introduction to computing programming. The CC2001-CS report offers various alternative implementation strategies to the "programming first" approach popularized by the definitions of CS1 and CS2 in *Curriculum '78*. These options include imperative-first, objects-first, or breadth-first in a 2-semester or 3-semester implementation, and algorithms-first, functional-first, or hardware-first in a 2-semester sequence.

CC2001-CS requires core "units" and "topics" be introduced from knowledge areas such as programming fundamentals (PF), programming languages (PL), and algorithms and complexity (AL). Depending upon the implementation strategy that is adopted, the PF, PL, and AL units and topics may be distributed among two or three introductory programming courses. Software engineering (SE) is usually covered later in the CS curricula, but some have recommended that SE be introduced at the beginning of the CS academic program (Cheston & Tremblay 2002). PSP serves to augment the SE component of the introductory course sequence by providing specific planning and quality management techniques to be used in program design, coding, and testing. However, the addition of PSP to the already expanding list of PF, PL, AL, and SE topics does increase the pressure to expand the CS programming course sequence from two to three semesters.

Many MIS programs have designed their programming courses according to IS 2002 (Gorgone, Davis, Valacich, Topi, Feinstein, & Longnecker 2002), the curriculum recommendations for undergraduate degree programs in information systems. In contrast to CC2001-CS however, IS 2002 recommends a single programming course, IS 2002.5 (Programming, Data, File, and Object Structures). The single course approach is partly the result of the IS focus on organizations and its systems rather than on algorithmic processes for information processing as with CS. However the scope of IS 2002.5 includes algorithm development, programming, and data structures in addition to software engineering methodologies.

The introductory version of PSP is a natural fit in such a course. PSP's time and quality management components complement the programming material, and provide structure from a software engineering perspective. But while the addition of PSP to IS 2002.5 has potentially the same benefits as in CS1 and CS2, it is perhaps doubly difficult to do so in a single semester. PSP may just be too much to add.

4 An MIS Database Course

The database course described in this paper is part of an MIS concentration in both undergraduate and graduate business programs in a School of Management. The undergraduate MIS concentration is part of an upper-division, AACSB-accredited BBA program. This program requires a lock-step sequence of 7 MIS courses, encompassing 4 semesters. The first semester includes a CS1 course (Java) taught by the CS department and a concurrent object-oriented analysis and design class taught by the MIS faculty. The remaining classes are all taught by MIS faculty, including a Java-based applications development class and a survey of hardware and software systems offered during the student's 2nd semester in the program. The database course is offered during the 3rd semester, as is a web-development class covering Java Script. The final semester of the MIS concentration includes a capstone development project.

The graduate MIS concentration requires that students complete a 1-semester programming course in any language (Java or C++ is recommended) prior to taking a 5-course MIS sequence within the scope of an AACSB-accredited MBA program. The MBA students are required to take the database course, object-oriented analysis and design, decision support systems, an IS project course, plus one MIS elective. All of these offerings are taught by MIS faculty, although the MIS elective course can be fulfilled by CS departmental offerings, assuming the student can satisfy graduate level CS course pre-requisites.

The BBA and MBA database courses are similar in content. The undergraduate students are actually more prepared for such a course via program prerequisites than are the graduate students. Both courses cover conceptual data modelling, database architectures, logical database design in the relational data model, data and database administration, physical database design, and data manipulation. The database textbook used during the Fall 2003 semester was Kroenke's *Database Processing* (2002).

Students complete two major projects in the course, including a database creation and manipulation project in MS Access, followed by replication of the project using Oracle SQL. Two exams are also scheduled. A data modelling exam covers conceptual modelling in Entity-Relationship and/or UML Class Diagram formats, logical database modelling in the relational data model, transformation of conceptual to logical models, and principles of data normalization. The 2nd exam requires that students code SQL statements for database creation, update, and manipulation involving queries of varying complexity.

5 PSP in the Database Course

PSP was introduced in both BBA and the MBA versions of the database course during the Fall 2003 semester. Components of PSP were introduced early in the semester through several lectures, coupled with concurrent coverage of introductory database management topics. Software development fundamentals were discussed

during the first 4 weeks of the semester, prior to any actual database development assignments. These topics included time management, period planning utilizing the PSP time recording log and weekly activity summaries, product planning utilizing the PSP job number log, and discussion of period vs. product planning. Database topics covered during that same period included an introduction to database management systems, input and output processing, file organizations, and the relational data model. During this period students applied the PSP time recording and planning techniques to basic course activities such as database text readings, completing PSP task assignments, and class lectures.

The 5th week of the semester marked the beginning of MS Access coverage (encompassing about 2 weeks of class lecture), at which time database development (or pseudo-programming activities) actually commenced. Course coverage of MS Access is limited to "tables" and "queries" objects, plus a brief introduction to "forms". There is no coverage of MS Access reports, macros, or modules. Table functions include table design, indexing, file import, and the relationships window to establish relationships and enforce referential integrity. In the MS Access project, students are required to create and load a database given a well-defined problem specification. They are then required to utilize query design view to formulate increasingly complex queries involving restriction, projection, inner and outer joins, product, summarization, and sorting. Students review the SQL code that MS Access generates, but do not code SQL statements directly.

Students are required to estimate the time they expect to expend on MS Access development activities before they begin the project, even though they may have little prior experience on which to base their estimates. These activities are partitioned into database definition, database loading, simple list queries, join queries, summarization queries, and miscellaneous queries. Each student tracks the actual time expended on each database development category on PSP time logs.

In addition to PSP time logs, students also record "size" measurements for the MS Access project. Software size is important to measure if it is in some way related to the effort expended in developing that software. If so, data on size and effort recorded during the development of one MS Access database can be used as the basis for project planning for future database development.

PSP normally utilizes a "lines of code" (LOC) measure for recording the size of a computer program (Humphrey 1997). However, such a measure is not applicable to MS Access since design forms are used rather than command-line coding. Thus, a counting standard was developed to measure the size of an MS Access database. The database definition task was measured in terms of number of tables, columns, certain column properties (e.g., formats, validation rules, or indexes applied), primary keys, foreign keys, and display controls specified. Database queries were measured in terms of the number of tables or sub-queries used, sorts, criteria rows, totals, and joins utilized. The objectives of this PSP assignment were to provide time and size baseline data for students to use in

estimating future MS Access development efforts, but also to establish baselines to be used in estimating Oracle SQL development activities for the same database.

The middle third of the semester focuses primarily on database topics, including conceptual data modelling (e.g., ERD's and UML Class Diagrams), CASE tools for conceptual modelling (e.g., Oracle Designer and Embarcadero Describe), logical database models, and principles of data normalization. Students are also completing the MS Access project during this period. PSP lectures are also given on project plans, software size measurement, and defect identification and tracking. These advanced PSP topics set the stage for more sophisticated project planning, monitoring, and analysis of the Oracle database project.

The final third of the semester concentrates on SQL, including implementation of a SQL-based database in Oracle. The Oracle project replicates the MS Access assignment, but requires SQL coding for the definition, update, and manipulation of the database. Some additional queries are also included, requiring SQL operations not readily available in MS Access query design mode. Two PSP assignments accompany the Oracle project. The first requires that students conduct project planning to estimate the size and effort anticipated in developing the SQL database. Students then track the time spent in each phase of development, and also record development size measured in terms of SQL LOC.

The second PSP assignment covers defect management. Students record the defects they find in the Oracle SQL specifications and queries that they code. PSP classifies defects into 10 general types applicable to most procedural or object-oriented programming languages. However these standard defect types are not necessarily appropriate for SQL. Thus, a SQL defect type standard was developed. The PSP and SQL defect types are summarized below in Table 2.

PSP Defects	SQL Defects
Documentation	Documentation
Syntax	Syntax
Build, package	Script Files
Assignment	Projection
Interface	Join, Product
Checking	Selection
	Set Operators
Data	Data
Function	Function
System	System
Environment	Environment

Table 2: PSP and SQL Defect Type Standard

Both PSP assignments were due at the same time as the Oracle SQL project.

6 Perceptions on PSP in the Database Course

6.1 Student Perceptions

Student perceptions concerning PSP can probably be summarized as simply "onerous". At least that's what the more constructive comments indicate. Unfortunately, the recording and tracking requirements imposed by PSP are tedious, perhaps more so in the database arena than in the programming realm. There are some automated LOC counters available for determining C++ or Java program sizes, but no such tools were available for counting MS Access database size. Students were also required to hand-tally SQL LOC to measure the Oracle database size, although text line counters could have been used. The preparation of time recording logs and defect logs are also labor-intensive, and hence unappealing to students.

Students did not generally see the "worth" in gathering such detailed data concerning their personal activities. PSP touts the benefits of accumulating personal data to be used in future project planning, but such usage requires at least several iterations of gathering data, analyzing it, then using it to estimate size and effort for the next program for such a discipline to be useful. While a single programming course, and certainly a 2-semester sequence, would typically entail at least several programs to which PSP is to be applied, the two database project approach did not provide a sufficient number of iterations of the PSP technique for its worth to be appreciated.

Finally, some students objected to lecture time being devoted to PSP material. Since the university's course catalog description characterizes the course as one in database management systems, a few students believed that the course should focus solely on that topic. Time spent in lectures on PSP meant less time for coverage of database material, particularly advanced database topics. PSP and software engineering in the context of database development were not as interesting to them as data warehouses, database administration, or object-oriented DBMS would have been.

6.2 Instructor Perceptions

The most notable aspect from an instructor's perspective is the time commitment required to integrate PSP into a course, whether programming or database oriented. The commitment embodies itself in pre-course preparation in PSP on the instructor's part, PSP lecture preparation and delivery, and instructor's time devoted during the course for monitoring, analysing, and grading student PSP data.

The instructor initially attended a 6-day faculty workshop (Cannon, Diaz-Herrera, Hilburn & Humphrey 2002) in preparation for teaching PSP (and its team-oriented TSP). Pre-requisite PSP readings and work were also required prior to the start of the workshop.

During the semester the instructor prepared several PSP lectures, a couple of PSP quizzes, and several PSP assignments. Much of these materials were adapted from the programming-course instructor materials distributed with the PSP textbook (Humphrey, 1997). However, modifications were necessary to adapt the materials to a

database context, including counting standards for MS Access and defect types for SQL, as noted earlier.

The most notable time commitment however, is required to monitor, record, and analyse the students' PSP data and to provide feedback to the students. Parts of this burdensome task could be delegated to a course graduate assistant (GA), although the instructor did not have a GA during the particular semester in question. The instructor also logged PSP time data in conjunction with teaching the course, as summarized in Figure 1.

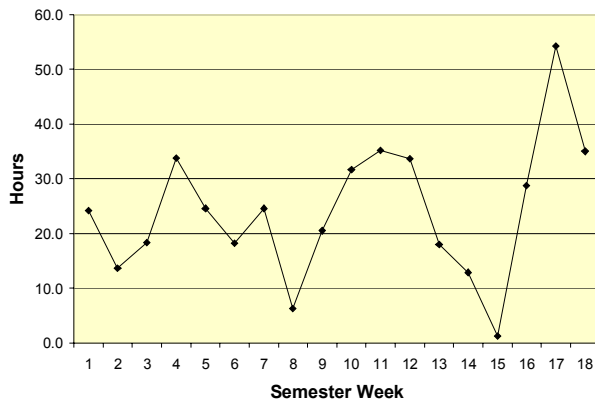


Figure 1: Instructor Workload in the Course

Figure 1 displays the number of hours the instructor spent teaching 1 graduate and 2 undergraduate sections (9 credit hours) of the database course during the semester in which PSP was introduced. The teaching task included classroom lecture time, class preparation, grading, and student consulting support. This data is for a well-developed course that the instructor has taught for a number of years. It generally requires only minor revisions in course content from year to year, with the exception of this instance in which PSP was added. One of the two "valleys" in the graph coincided with the university's fall break (week 8). The other "valley" (week 15) occurred during the U.S.A.'s Thanksgiving holiday break (week 15), during which the instructor also presented a paper at a professional conference and hence taught no classes that week. The mean time spent on the course per week was 24.2 hours (26.7 hours if weeks 8 and 15 are excluded).

It should be noted that PSP time logs are used to record *only* direct time devoted to project tasks. Casual interruptions of even a couple of minutes are excluded from the time log (e.g., taking a coffee break, answering the phone, or reviewing email). Thus, for example classroom lecture time is logged as 7.5 hours per week for 9-credit hours of coursework since 2 sections met Tuesday/Thursday for 1:15 for each class period, and 1 section met on Thursday for 2:30. The time spent walking to class, setting up the classroom, cleaning up afterwards, and chatting with students after class is not logged as task time in Figure 1.

PSP and TSP project experience has indicated that for a standard 40-hour workweek in the U.S.A., 20 hours is an aggressive goal for task-related work (Webb & Humphrey 1999). Some anecdotal discussions between the instructor and project managers at a local organization

using PSP/TSP suggest that 15 hours per week is more realistic. Thus, the 25 hours per week logged by the instructor in the database course throughout the semester represents a full workweek by any standard. That did not leave much time for the instructor's other faculty obligations in research and service.

The instructor was disappointed that students did not value the PSP discipline more highly. While the application of PSP is admittedly tedious, industry studies (Hayes & Over 1997; Ferguson, Humphrey, Khajeroori, Macke, & Matvya 1997) have demonstrated clear benefits. Perhaps the instructor needs to undertake a more extensive marketing campaign to "sell" PSP to the students. If so, this marketing campaign should utilize the students' own PSP data to the fullest extent possible, within the data processing constraints represented by the volume of data submitted.

PSP clearly requires a progressively more detailed and repetitive application of its recording and tracking activities for software development. Thus, just as in a programming course in which several more progressively difficult programs will be assigned, database development must be partitioned accordingly into several activities or assignments. As noted earlier, the MS Access and Oracle projects were partitioned into several parts. However, the parts were combined for submission purposes into two basic projects. A better approach would have involved the break-down of each project into a series of sub-assignments, due one-at-a-time in stages, to take advantage of PSP's cyclic development approach. The repetition of assignment, submission, and feedback is critical in order to illustrate PSP's benefits.

7 Conclusions

PSP has an important role to play in the database course. The structure and discipline represented by PSP can serve to guide students in the application of database theory and database language constructs to the database development process. Software engineering fundamentals underlying PSP provide an overall perspective to database development that is sometimes lacking in the database course. The specific techniques for project planning and defect management in PSP also provide students with an excellent opportunity to learn more about their personal development practices. This information can prove useful in later personal and project planning endeavours in both the classroom and the workplace.

Some adaptations are necessary in order to integrate PSP into the database course. PSP must be adapted to reflect database development "language" constructs, encompassing "form" based specification such as encountered in MS Access design views, as well as "command-line" languages such as SQL. Likewise, the database content (or at least the assignments) must also be adapted to conform to the repetitive, cyclic approach to software development inherent in PSP. Database projects must be partitioned in order to provide several development cycles during the semester to which PSP can be applied. Through modifications of both types, PSP can be successfully integrated into the database course.

8 References

- ACM Curriculum Committee on Computer Science (1979): Curriculum '78: Recommendations for the Undergraduate Program in Computer Science. *Communications of the ACM* **22**(3): 147-166.
- Borstler, J., Carrington, D., Hislop, G.W., Lisack, S, Olson, K. & Williams, L. (2002): Teaching PSP: Challenges and Lessons Learned. *IEEE Software* **19**(5) 42-48.
- Cannon, R., Diaz-Herrera, J., Hilburn, T., & Humphrey, W.S. (2002): Teaching a Software Team Project Course: Improve Team Performance by Teaching Engineering to Teams. Carnegie Mellon Software Engineering Institute Summer Faculty Workshop, Southern Polytechnic State University, Marietta, GA, USA, July 8-13, 2002.
- Capability Maturity Model Integration for Software Engineering (CMMI-SW) Version 1.1: Staged Representation (2002). CMU/SEI-2002-TR-029.
- Cheston, G.A., & Tremblay, J-P. (2002): Integrating Software Engineering in Introductory Computing Courses. *IEEE Software* **19**(5) 64-71.
- Joint IEEE Computer Society/ACM Task Force (2001): Computing Curricula 2001: Computer Science. <http://www.computer.org/education/cc2001/>. Accessed 12 Aug 2003.
- Ferguson, P., Humphrey, W.W., Khajeroori, S., Macke, S., & Matvy, A. (1997): Introducing the Personal Software Process: Three Industry Case Studies. *IEEE Computer* **30**(5),24-31.
- Gorgone, J.T., Davis, G.B., Valacich, J.S., Topi, H., Feinstein, D.L., and Longnecker, H.E., Jr. (2002): IS 2002 Model Curriculum and Guidelines for