Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1997 Proceedings

Americas Conference on Information Systems (AMCIS)

8-15-1997

Applied Software Process Improvement

Rick Gibson
American University

Follow this and additional works at: http://aisel.aisnet.org/amcis1997

Recommended Citation

Gibson, Rick, "Applied Software Process Improvement" (1997). *AMCIS 1997 Proceedings*. 55.
http://aisel.aisnet.org/amcis1997/55

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 1997 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Applied Software Process Improvement

Rick Gibson
Department of Computer Science and Information Systems
American University
Washington, D.C.

Introduction

Increasing use of the Software Engineering Institute (SEI) Capability Maturity Model (CMM) has been accompanied by debate regarding its relevance (see Schwaber, 1996 or Bach, 1997). Consensus has emerged concerning a key CMM principle--the quality of a software product is governed by the quality of the process used in development. However, conflict centered on the determination of a relevant definition of process remains. Related questions central to the debate focus on whether the CMM applies only to large organizations. CMM success in helping organizations improve their process has resulted in its adoption by most major Department of Defense (DoD) contractors as a process improvement framework, many of which are now at or above CMM Level 2 maturity.

Although these CMM improvement efforts have been beneficial, they have predominantly addressed management issues. As firms approach and move beyond Level 3, they find that further maturity is dependent on improving individual performance. Regardless of how well an organization is managed, the quality of software work is governed by the practices of the software engineers. Data show that even experienced software engineers inject about 100 defects into every 1000 lines of code written. Typically, 25 of these defects remain undiscovered until final system testing, at which time each defect costs several thousand dollars to fix. Jones (1991) suggests that the defect potential of an application to be the number of function points raised to the 1.2 power--highlighting the significance of the defect problem for the larger systems that are becoming ever more commonplace.

In response Humphrey (1995) researched the applicability of the CMM concepts to the smallest of groups--individual programmers--to develop a personal software process (PSP^{SM}). As with the CMM, the key PSP principles are indisputable: the quality of a software product is governed by the quality of its worst component-which in turn is governed by the individual who developed it. One PSP objective is to encourage individual software engineers to apply proven quality principles such as eliminating all defects before testing.

Humphrey (1996b) contends that there are two serious consequences of not training software engineers to consistently produce quality work. First, since only the software engineers know the details needed for accurate plans, they need to be trained to make plans and routinely estimate, track, and report on their work. Second, high-quality software systems can only be produced when every engineer produces high-quality component parts. Any single engineer using poor practices results in extending overall system testing time and a poor-quality product is the likely outcome. To produce quality products, all software engineers must adopt process improvements that use effective quality practices, establish personal quality goals, and track their progress against these goals.

The PSP is a scaled-down version of a state-of-the-art industrial software process that is suitable for individual use. In order for software engineers to accept and use process improvement techniques, they must first believe them to be effective. Realizing that software engineers would demand data in support his assertions, Humphrey initially wrote a textbook and presented the PSP to computer science students. Student data combined with later results from seven industrial projects showed that PSP-trained software engineers completed projects on or ahead of schedule, and none of the industrial products had any defects reported during several months of customer use (Ferguson, et al. 1997).

PSP Applied In Software Education

Most software engineers learn to write programs in courses that focus on effective, efficient problem-solving but ignore discussion of plans, quality, or metrics. Not surprisingly, upon graduation they apply the accepted practice of writing code of unknown quality and rely on compiling and testing to find and fix the defects. In contrast, Khajenoori (1994) describes a new Masters of Software Engineering program at Embry- Riddle Aeronautical University that incorporates process concepts, including the PSP as a means to introduce engineering discipline into software development. The PSP is data-driven: measurements and statistical techniques highlight process deficiencies and provide a focus for improvement. By providing a framework for data collection using assignment kits, PSP helps individuals develop a quantitative understanding of their process steps. It also demonstrates how process methods can improve quality and productivity. Evaluation of the new program is on-going, but reports are favorable from other students and faculty who, as a result of PSP, are more aware of rigor, process and variety.

Several formal methods are used in the PSP. In planning, engineers follow a proxy-based estimation (PROBE) method and use their personal data and linear regression to make statistically sound estimates. Size proxies in PSP are objects and methods that enable engineers to effectively visualize program size. They also learn how to derive a prediction interval as a measure of the accuracy of their estimates. Schedule and task planning templates are introduced, and projects are tracked with the earned-value method, that helps determine project status and predict project completion. Planning accuracy is measured by a cost-performance index, which is a ratio of planned to actual development cost.

The PSP measures are defined using the Goal-Question-Metric paradigm of Basili and Weiss (1984). With PSP quality management, engineers track their own defects, find their defect removal yields, and calculate cost-of-quality (COQ) measures. Further comment on quality is pertinent-while the COQ metric is traditionally associated with organizational efforts, the PSP emphasizes tracking *individual* COQ with three components: failure costs (diagnose, repair, re-test) appraisal cost(reviews and inspections), and prevention costs (identification of causes). Pareto defect analysis is used to derive personal design and code review checklists, which the engineers update with defect data from each new project. The PSP also addresses design quality, another potential area of defect prevention. Traditional software design methods focus on producing designs but fail to specify exactly what a design is and what it must contain when completed. Without this specification it is difficult to perform effective design reviews or to evaluate and improve design quality.

Results of PSP Applications

Humphrey (1996) reports that a substantial number of industry and government groups currently use or are introducing the PSP. The SEI is working with many leading software organizations in both DoD and commercial industry. Some of the leading corporations currently introducing the PSP include: Advanced Information Systems, Citicorp, Ford Motor Co., Harris, Motorola, and Union Switch and Signal. As a result, data on their experiences are now becoming available.

As an example of these findings, 104 engineers at Advanced Information Systems in Peoria, IL who completed the ten PSP course exercises reduced the interquartile range of the estimating errors by 40 percent. Ferguson et al. (1997) provide further examples. On one development project three engineers guessed how long it would take to develop the first three components of a large product. Their initial estimating errors averaged 394 percent. One engineer estimated that his work would take six weeks when it took 26 weeks. Management stopped work, had the engineers trained in the PSP, then had them replan the project. After PSP training, these same engineers' estimates had an average error of -10.4 percent; that is, they finished an average of 10.4 percent early. A key to estimating success with the PSP is that engineers use actual historical data to make plans. By comparing actual results with plans, they can see their mistakes and are more likely to make better plans in the future.

The reported PSP quality results are equally significant. The number of defects that engineers found in their programs was sharply reduced by the PSP course. For the 104 engineers, average total defects were reduced by 58 percent, average compile defects were cut by 83 percent, and average test defects were 72 percent lower. Schedule and quality performance data on six projects at Advanced Information Services are provided in Tables 1 and 2.

Table 1: Project Schedule Results

Project	Staffing PSP/Non PSP	Plan Delivery	Actual Delivery
A	3/0	7 months	5 months
B	0/3	2 months	5 months
C	0/3	10 months	19 months
D	1/0	6 months	6 months
E	1/0	2 months	2 months
F	2/1	2 months	2 months

Table 2: Project Quality Results

Project	Staffing PSP/Non PSP	Acceptance Test Defects	Usage Defects
A	3/0	1	0
B	0/3	11	1
C	0/3	6	14
D	1/0	0	0
E	1/0	0	0
F	2/1	0	3

Introducing the PSP

Considering the SEI's reports regarding PSP introduction experience, Humphrey (1996b) suggests the following steps for applying the PSP as an introduction to software process improvement:

1. Develop an introduction plan and strategy.

2. Management reviews the costs and benefits of PSP and publicly supports the introduction plan.

3. Training is provided to the project team, and the supervisor is trained along with the team.

4. Management views the PSP course and exercises as part of the job and tracks progress with the same priority as a development project.

5. Following training, the PSP methods are immediately applied to a project.

6. Directly involved line managers are familiar with and support the PSP strategy and have been trained in how to manage PSP-trained engineers.

With qualified instructors, training takes about 125 hours per engineer. Engineers should be encouraged to complete the full PSP course in order to acquire the personal data needed to convince themselves of the benefits of the PSP methods. The SEI offers instructor training for organizations that wish to train their employees themselves.

Conclusion

The CMM practices have been proposed, debated, applied and reviewed for the past several years by thousands of software engineers. Documented evidence exists for organizational benefits ranging from productivity to defect prevention, schedule improvement and return on investment (for more see Paulk et al., 1995). It is reasonable to conclude that the PSP will provide comparable success at the individual level.

Yourdon (1997) concluded his review of Humphrey's book on PSP in this manner "….if you believe that software development is something more serious than extemporaneous coding in Visual Basic or PowerBuilder, then Humphrey's new book is one that you need to understand, and whose advice and guidance you need to follow."

References

Bach, J. (1997). The Hard Road From Methods to Practice. *IEEE Computer* (February):129-130.

Basili, V.R. and Weiss, D.M. (1984). A Methodology For Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, (November):728-738

Humphrey, W.S. (1995) *A Discipline For Software Engineering.* Reading, MA: Addison-Wesley.

Humphrey, W.S. (1996). Using a Defined and Measured Personal Software Process. *IEEE Software*, (May):77-88.

Humphrey, W.S. (1996b). Making Software Manageable. *Crosstalk: The Journal of Defense Software Engineering*, (December):3-6.

Jones, C. (1991). *Applied Software Measurement*, 2nd Edition, New York: McGraw-Hill.

Khajenoori, S. (1994). Process-Oriented Software Education. *IEEE Software* (November):99-101.

Paulk, M, Weber, C.V., Curtis, B., and Chrissis, M.B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*, Reading, MA: Addison-Wesley.

Pierce, K.R. (1993). Rethinking Academia's Conventional Wisdom. *IEEE Software*, (March):94-99.

Schwaber, K. (1996). Defining Process vs. Problem-Solving. *Application Development Trends*, (March):76-82.

Shaw, M. (1990). Seeking A Foundation For Software Engineering. Interview by W. Myers, *IEEE Software* (March):102-103

Wasserman, A.I. (1996). Toward a Discipline of Software Engineering. *IEEE Software*, (November): 23-31.

Yourdon, E. (1996) Review of A Discipline for Software Engineering. *http://www.yourdon/504Humphrey.html*