The Personal Software Process: Downscaling the factory?

56-61

Daniel M. Roy
Software Technology, Process and People (STPP)
20 Forest Rd. Bradford Woods, PA 15015
(412) 934 0943 E-mail: dmr@sei.cmu.edu
(Visiting scientist, SEI)

Abstract: It is argued that the next wave of software process improvement (SPI) activities will be based on a People-centered paradigm. The most promising such paradigm, Watts Humphrey's Personal Software Process (PSP) is summarized and its advantages are listed. The concepts of the PSP are shown to also fit a down-scaled version of Basili's experience factory. The author's data and lessons learned while practicing the PSP are presented along with personal experience, observations and advice from the perspective of a consultant and teacher for the Personal Software Process.

1 Toward a People-centered SPI paradigm

The Capability Maturity Model (CMM) and CMM-based SPI paradigms have had a profound impact on the organizational practices within the software industry [Herbsleb-94]. Other SPI paradigms such as the experience factory have been demonstrating the value of experiment based software improvement for over 15 years [IEEE-94]. In spite of these progress, we technologists, process advocates and other change agents still have to fight an entrenched and pernicious resistance.

To better ascertain what to do about this, we must understand where we have been and where we want to go next. As Basili puts it in [Basili-89]:

> *'We have evolved from focusing on the project, e.g. schedule and resource allocation concerns, to focusing on the product, e.g. reliability and maintenance concerns, to focusing on the process, e.g. improved methods and process models'*

However, addressing the practitioner's resistance from healthy skepticism to outright obscurantism is not a technical problem; it is a human concern. Perhaps accelerated progress requires that we now continue the evolution by focusing on the People, e.g. individual education and practices based on individual self improvement.

Major relatively new concepts such as the CMM or the experience factory are both intellectually satisfying and daunting to practice at the individual level. As a programmer, I may well understand the importance of the practices of the subcontract management KPA while at the same time failing to relate to any of them in my individual work. As a reuse technologist, I may be totally convinced that my company should operate as an experience factory while at the same time having no idea how to incorporate the concepts in my day to day practice.

Conversely, I may be highly skeptical of 'their' SCEs, 'their' pilot project, and God knows what other latest fad. I will remain unconvinced until 'they' show me that it will really work for me. I may have heard good things about clean room, I may even have watched a convincing presentation at the Software Engineering Workshop about it. If I have never personally experienced it, it will remain alien to me, something even vaguely frightening that I will keep resisting. In the word of a most famous (and anonymous) Chinese proverb:

> *'I hear and I forget, I see and I remember, I do and I understand'*

A more personal and more practical approach to software improvement where the individual practitioner learn by doing, may be needed to accelerate the transition of better engineering practices throughout our organizations.

2 The Personal Software Process

As students, we typically practice on toy problems in programming language classes. Our ad hoc processes are sufficient to produce moderate size programs quickly and get a passing grade. As programmers we quickly discover that these student practices do not scale up but what can we do?. The product must be out the door if we want to work on the next one. There is very little time to experiment with something unproven.

The personal software process was developed by Watts Humphrey to indoctrinate students (in university and industry alike) in the use of large scale methods based on the CMM. To quote Watts in [Humphrey-95], the PSP...

> *'... scales down industrial software practices to fit the needs of small scale program development. It then walks you through a progressive sequence of software processes that provide a sound foundation for large-scale software development'*

Using fairly simple and well proven engineering principles, the PSP student plans his work, enacts a well defined process, building the product while gathering data, and performs a post mortem that seeds the next improvement cycle. This personal approach to software improvement offers the following advantages:

- By having to adhere to more disciplined practices, students learn a lot about process, engineering, and software improvement. Most becomemotivatedto learn even more about their field
- By gathering their own private data, students quickly build a significant experience base which allow them to set new goals, perform the next experiment and check the results against the goals
- Since the data is personal and private, PSP practitioners need no convincing from anyone about the value of a process step or a technology. They know whether it works for them or not based on their own quantitative results
- Armed with their own productivity and quality statistics, practitioners of the PSP are better able to make commitments they can meet. They can also better resist unreasonable commitment pressures

The PSP course leads the student to the gradual application of software engineering discipline through a set of 10 assignments:

1. Average and standard deviation using linked list
2. Physical line counter
3. Object LOC counter (build on 2)
4. Linear regression using linked list (build on 1)
5. Standard distribution (integration by the method of Simpson)
6. Linear correlation (build on 5)

7. Confidence intervals (build on 5 & 6)
8. Sorting a set of numbers in ascending order
9. Performing statistical fit tests on the above data
10. Computing muti-linear regression coefficients (by solving a system of linear equations)

These simple exercises were found to have the following advantages:

- Simplicity without being trivial.
- Fostering reuse and good object oriented development practices
- Gradually building a small PSP support toolset

The PSP data shown on the transparencies was collected during Watts Humphrey's Spring 94 course for the Master of Software Engineering at CMU.

3 Personal data, experience, and lessons learned

Several PSP reports have to be written as part of the course detailing:

- Evolution of size and time estimates accuracy
- Pareto charts and checklist for defects
- Defect injection and removal trends
- Cost per defect type and injection/removal phases
- Process development process for PSP reports
- Detailed process analysis such as A/F ratio
- Lessons learned
- Future steps

The large number of graphs could not be reproduced here or even shown during the talk. Watts Humphrey's data analysis diskette (which can be obtained with the book [Humphrey-95]) includes an optimum set of Excel templates and macros for PSP data analysis. I found it very useful to track my progress and accelerate the routine of the post mortem analysis.

A central part of my talk dealt with the application of the concepts of experience factory to my PSP results. The experience gathered can be summarized as follow:

- The accuracy of my time and size estimates improved from +-40% to +-20% over the 10 assignments of the PSP.
- The PSP linear regression model helped me increase the accuracy of my size estimates. The multi linear coefficients computed by program 10 offer great potential to similarly increase the accuracy of my time estimates.
- The percentage of development time spent compiling decreased from 15% to 5%.
- My productivity during the development phase remained at 20 LOC/hr.
- I made a humiliating number of syntax errors with a language I know well until I truly inspected my code BEFORE compiling it.
- My error injection rate decreased from 180/kLOC to 30/kLOC and from 4 defect/hr to less than 1 defect/hr.
- From assignment 4 on, the sum of my code reuse and code developed for reuse stayed at about 80%.
- Defect fix cost varied from 1 min/defect to 8 min/defect depending on phase injected/removed.
- The process development process I enacted to develop a report development process for my PSP experience reports was an overkill. But I learned a lot trying that hard.

Building on this experience, I have applied the GQM paradigm to the definition of my next process improvement steps:

- Reduce my error injection rate to less than 20 defects/kLOC
- Improve my error detection processes
    - Keep design and code inspection yields above 50%
    - Keep formal pre-compile inspection yield above 80%
    - Strive for zero compile error
    - Improve my testing process to a yield over 50%
- Keep containing costs
    - Keep personal and informal review rates above 200 LOC/hr
    - Keep formal inspection rates above 100 LOC/hr
- Increase reuse
    - Either assemble 80% of the software out of reusable components
    - Or make reusable components out of at least 50% of the new code
- Formalize the experience gathered with the PSP by applying experience factory concepts

4 Teaching the PSP

SEI has already conducted one 'Train the trainer' course In Pittsburgh from October to December 1994. I taught the 2 lectures on design in that occasion. Besides the usual lessons learned from our own lectures, I think all instructors agreed that:

- The PSP is not your usual "teach and run" course
- Serious commitment is necessary from both student and sponsor
- A qualified instructor is necessary to get long term results

The PSP is about behavioral change. It is not a typical lecture course. It is a 200 hr intensive educational experience. The lectures are but the tip of the iceberg. The instructor must spend a significant amount of time tutoring the student in the correct implementation of the organization's process. The students don't just sit there either, they write working programs. These programs have to be reviewed and corrected. The process must be analyzed and feedback must be given. The PSP is more like a complete training program (in the sense of the CMM level 2 KPA) and typically spans 20 weeks. Strong commitments are necessary:

- from the student to honestly work the exercises, improve his process, and to finish the course
- from the sponsor to allow the time necessary for the lectures and for part of the implementation of the programs (typically shared 50/50 between sponsor and student)

Best resuts are seen when the sponsor treats the PSP assignment as any other (assuming correct project tracking and oversight practices). This means that the student's assignments are integral part of the workday and are part of his deliverables.

The PSP also requires a dedicated and qualified instructor with demonstrated programming and software management experience. Based on historical data, the effort necessary to correctly teach the PSP is roughly:

- Lecture preparation: 2-4hrs/lecture
- Tutoring: 5-10 hrs/student
- Program & process analysis: 2-10 hrs/student

Anything less has a great chance of failing to make a lasting difference in the disciplined, quality-driven individual practices demonstrated in the PSP course.

5 Conclusions

The PSP gives me the opportunity to improve the quality of the software I produce by offering a framework for objective measurement and improvement of my practices. However, the accuracy and the consistency of the data gathering process is paramount. Watts made this point very clear throughout the course. Nevertheless, it took me quite a while to truly understand why. I believe that a strict data inspection process should be enacted and particularly strongly enforced at the beginning of a PSP course to ensure that all students start on the right foot. I also believe that the postmortem phase should be expanded to include the systematic analysis and archiving of lessons learned with the assignment at hand. I have modified my own PSP accordingly.

I believe that the PSP is not only about scaling down the CMM. It can also be seen as a scaled down experience factory. It is because the PSP encompasses such an elegant synthesis of large scale methods that it will power the next wave of software practice improvement.

By practicing the PSP, I have learned a great deal about enacting, improving and even developing personal processes. I have carried the very simple principles of the PSP and the process development methodology described in chapter 13 of [Humphrey-95] to other processes:

- The organization of my work day
- A consulting personal process
- A process to perform Rate Monotonic Analysis
- A family of processes to write papers and reports.

These have been very exciting first steps.

6 Bibliography

[Basili-94] Victor Basili et. al., 'The Experimental Paradigm in Software Engineering', Experimental Software Engineering issues, Springer-Verlag, 1994.

[Herbsleb-94] James D. Herbsleb and David Zubrow, 'Software Process Improvement: An Analysis of Assessment Data and Outcomes', Technical report CMU/SEI-94-TR7, September 1994.

[IEEE-94] IEEE, 'IEEE Computer Society Award for Software Process Achievement, Nomination of 1994 Award Winner', Information bulletin, May 1994.

[Park-92] Robert E. Park, 'Software Size Measurement: A Framework for Counting Source Statements', Technical report CMU/SEI-92-TR-20, September 1992.

[Humphrey-95] Watts S. Humphrey, 'A discipline for Software Engineering', Addison Wesley, January 1995.

Carnegie Mellon University
Software Engineering Institute

The PSP: Downscaling the factory?

Daniel M. Roy, SEL workshop, December 1994

Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213

Sponsored by the U.S. Department of Defense

Carnegie Mellon University
Software Engineering Institute

Agenda[1]

The Personal Software Process (PSP)

Some preliminary results

SEL experience factory

Scaling down the models: The experience workshop

1

[1] Preliminary work offered for informal review.

Carnegie Mellon University
Software Engineering Institute

The Personal Software Process

Programming language class practices do not scale up

Corporate wide efforts encounter increasing resistance on the way down.

The PSP:

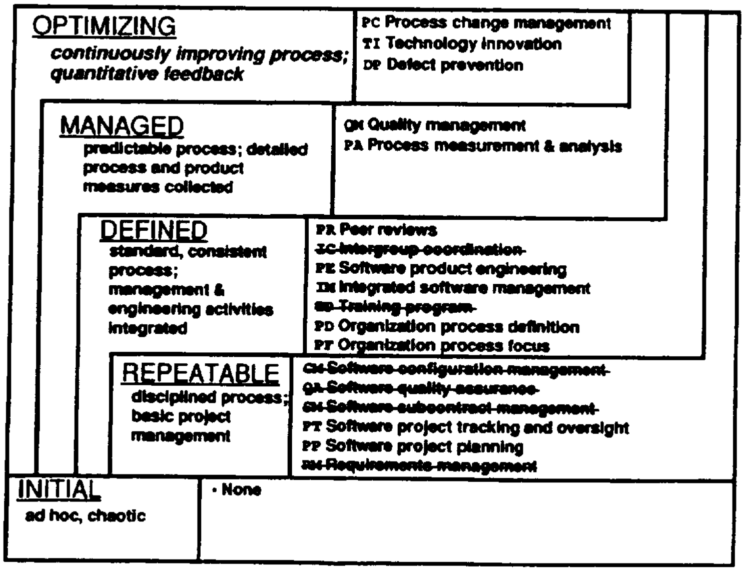
> *"Scales down industrial software practices to fit the needs of small scale program development. It then walks you through a progressive sequence of software processes that provide a sound foundation for large-scale software development."*[1]

[1] Watts Humphrey, "A discipline of software engineering", Addison Wesley, December 1994.

2

Carnegie Mellon University
Software Engineering Institute

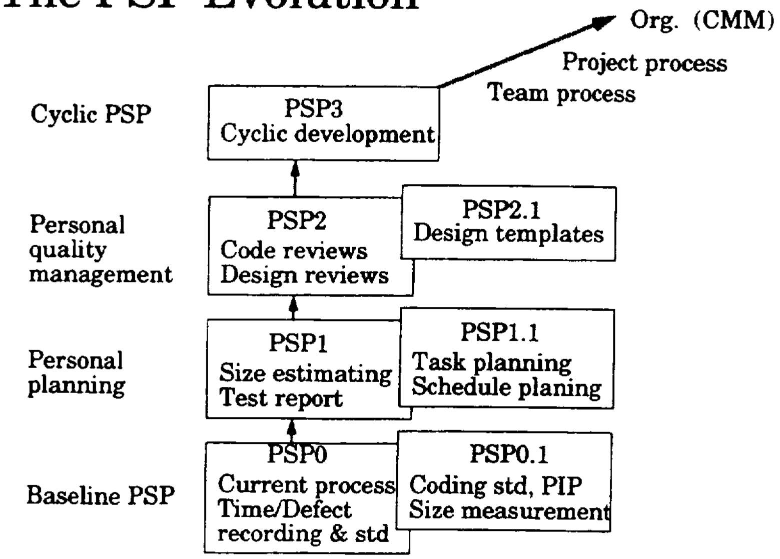
KPAs scaled down for the PSP



OPTIMIZING *continuously improving process; quantitative feedback*	PC Process change management TI Technology innovation DP Defect prevention
MANAGED predictable process; detailed process and product measures collected	QM Quality management PA Process measurement & analysis
DEFINED standard, consistent process; management & engineering activities integrated	PR Peer reviews ~~IC Intergroup coordination~~ PE Software product engineering IM Integrated software management ~~SP Training program~~ PD Organization process definition PF Organization process focus
REPEATABLE disciplined process; basic project management	~~CM Software configuration management~~ ~~QA Software quality assurance~~ ~~SM Software subcontract management~~ PT Software project tracking and oversight PP Software project planning ~~RM Requirements management~~
INITIAL ad hoc, chaotic	• None

2

The PSP Evolution[1]



Org. (CMM)
Project process
Team process

Cyclic PSP

PSP3
Cyclic development

Personal quality management

PSP2
Code reviews
Design reviews

PSP2.1
Design templates

Personal planning

PSP1
Size estimating
Test report

PSP1.1
Task planning
Schedule planing

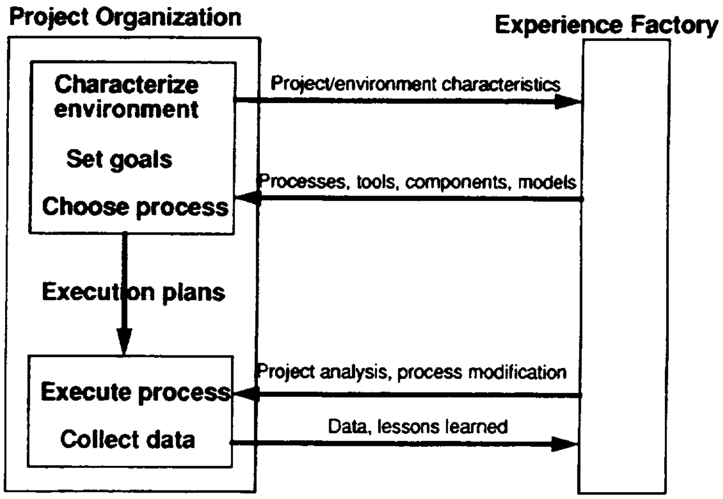
Baseline PSP

PSP0
Current process
Time/Defect recording & std

PSP0.1
Coding std, PIP
Size measurement

[1] Watts Humphrey, "A discipline of software engineering", Addison Wesley, December 1994.

The Experience Factory context[1]



Project Organization

Experience Factory

Characterize environment

Set goals

Choose process

Execution plans

Execute process

Collect data

Project/environment characteristics

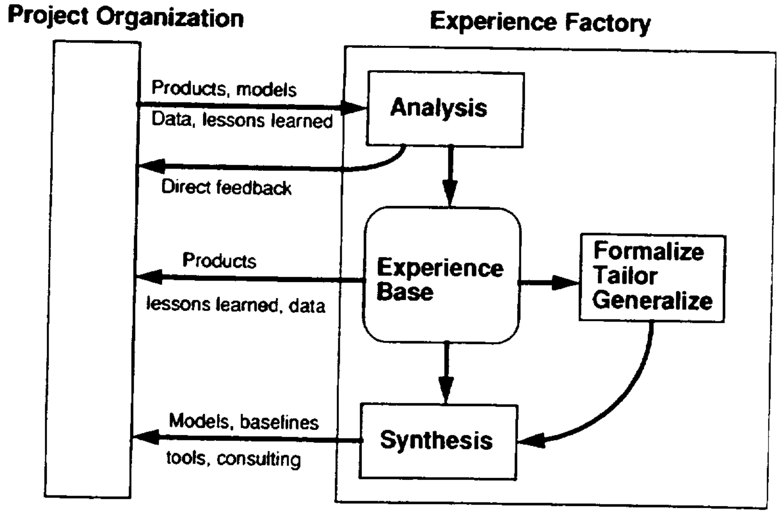
Processes, tools, components, models

Project analysis, process modification

Data, lessons learned

[1] From "The Experimental Paradigm in Software Engineering, Experimental Software Engineering issues", Rombach, Basili, Selby, Springer-Verlag

The Experience Factory structure[1]



Project Organization

Experience Factory

Products, models

Data, lessons learned

Analysis

Direct feedback

Products

lessons learned, data

Experience Base

Formalize Tailor Generalize

Models, baselines

tools, consulting

Synthesis

[1] From "The Experimental Paradigm in Software Engineering, Experimental Software Engineering issues", Rombach, Basili, Selby, Springer-Verlag

The PSP assignments as experiments

Goal:

- **Actual staff-hours will be within 20% of estimates 80% of the time.**

Questions:

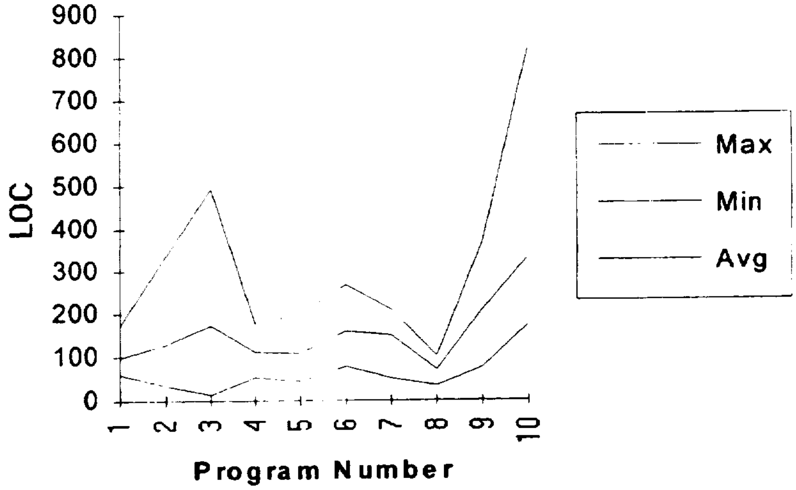
- **How do I predict my effort now?**
- **How do I measure the actual effort?**
- **How do I track actual against estimates?**
- **What is the dispersion now?**
- **If I had a data base of these, I could get statistics**

Metrics:

- **Estimate in mn before, measure actuals during**
- **Compute linear regression and confidence intervals from the data base. Accuracy is given by stats.**

Carnegie Mellon University
Software Engineering Institute

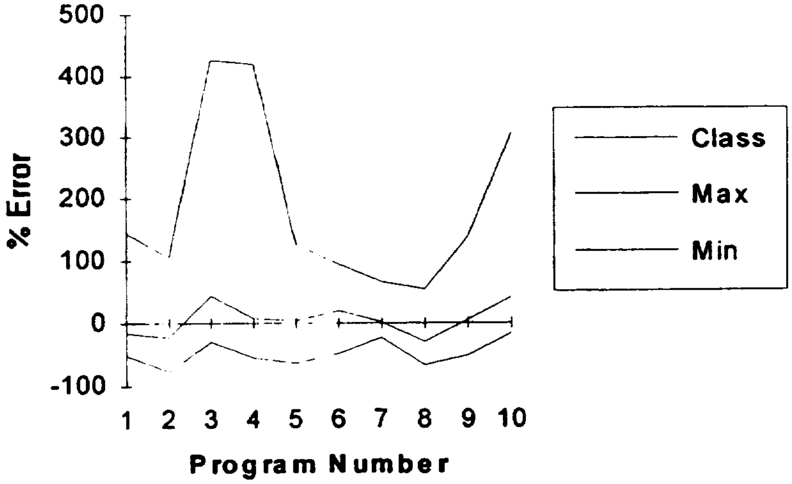
Actual Size Range



9

Carnegie Mellon University
Software Engineering Institute

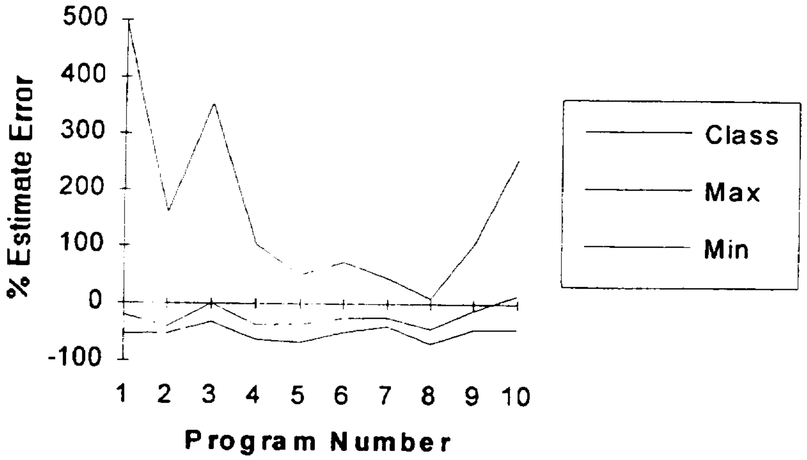
Size Estimating Error Range



10

Carnegie Mellon University
Software Engineering Institute

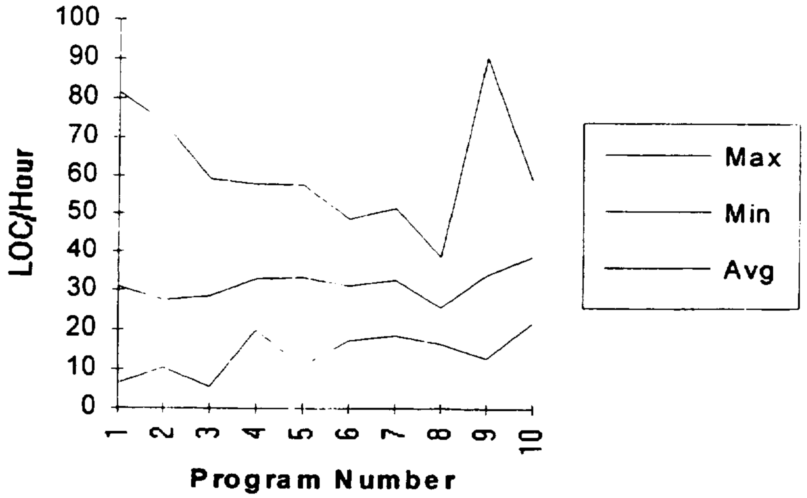
Time Estimating Accuracy - % Error



6

Carnegie Mellon University
Software Engineering Institute

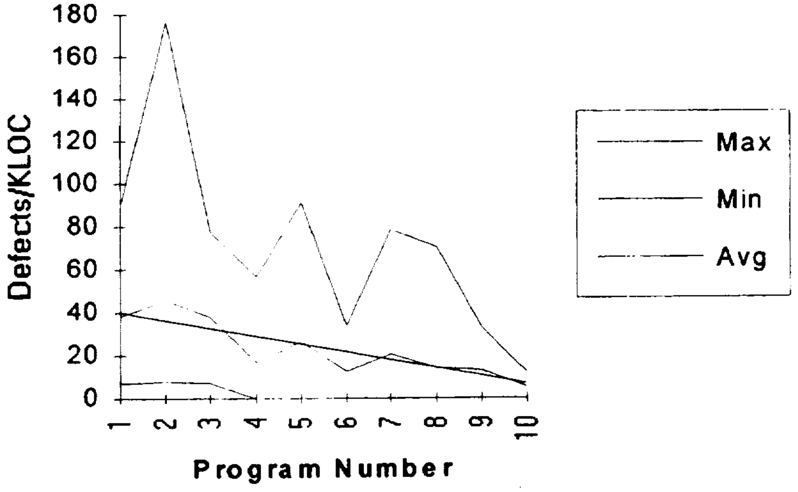
Productivity Range



21

Carnegie Mellon University
Software Engineering Institute

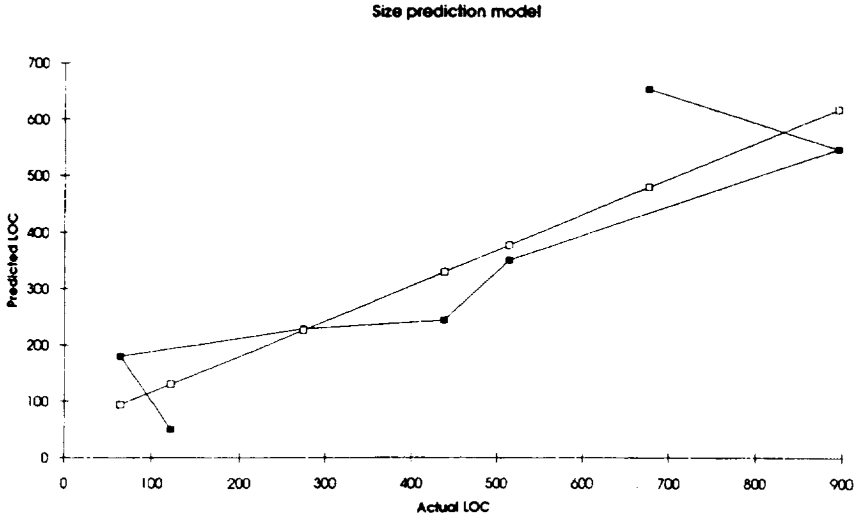
Defects Found in Test - Range



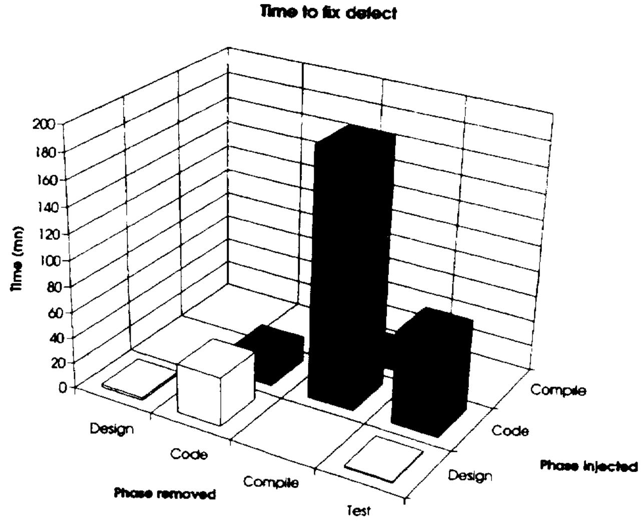
20

Carnegie Mellon University
Software Engineering Institute

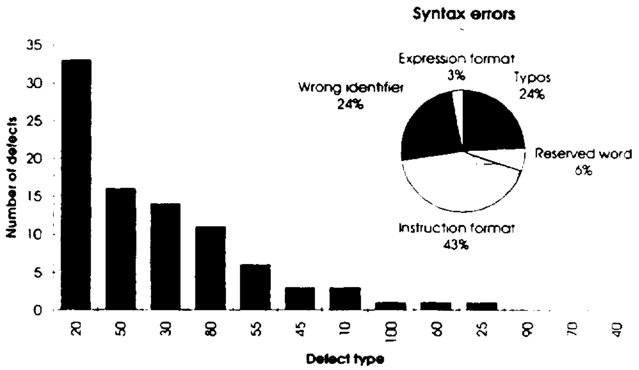
Size prediction model (dmr data)



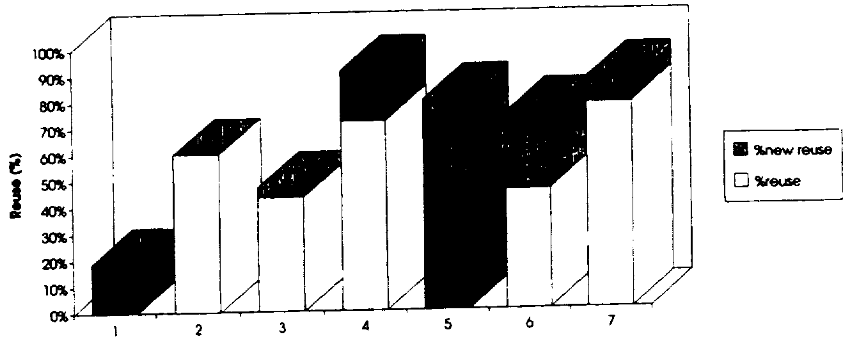
Cost of error (dmr data)



Defects analysis (dmr data)



Reuse trends (dmr data)



Ada PSP: Some experience artifacts

> *"I hear and I forget, I see and I remember, I do and I understand"*[1]

A lot of very useful process data:

- **predicted and actual time per phase**
- **error classes and distribution**
- **linear regression models for size and cost estimates**
- **trend analysis graphs on all of the above**
- **post mortems and reports as experience base**
- **a deeper understanding of PSI that carries beyond software development**

A lot of new goals and ideas to try next

[1] Anonymous Chinese proverb

Some of my next goals:

Reduce my total defect injection rate to less than 20 per KLOC.

Optimize my set of inspection processes to reduce their cost to less than 1 inspection staff-mn per SLOC while keeping yield above 80%

Either build with reuse (at least 80% of total SLOC) or build for reuse (at least 50% of the new code is reusable)

Revisit the PSP in the light of the CMM and ISO 9000-3

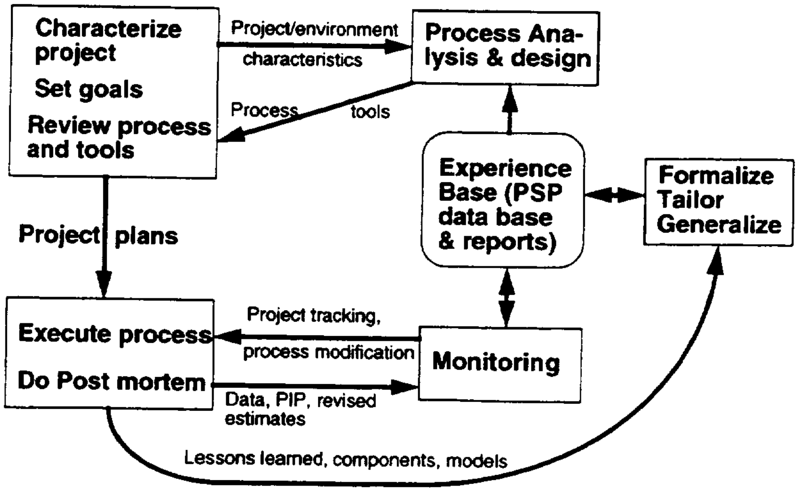
Recast the PSP in the experience factory mold

13

PSP: The experience workshop

Plan/Do

Check/Act



Characterize project
Set goals
Review process and tools

Project/environment characteristics

Process Analysis & design

Process tools

Experience Base (PSP data base & reports)

Formalize
Tailor
Generalize

Project plans

Execute process
Do Post mortem

Project tracking, process modification

Data, PIP, revised estimates

Monitoring

Lessons learned, components, models

14

Conclusion

The PSP represents an elegant synthesis of proven concepts (CMM, experience factory) scaled down to the individual level.

Preliminary PSP results are encouraging. Team data is needed.

Until now:

> *"We have evolved from focusing on the project, e.g. schedule and resource allocation concerns, to focusing on the product, e.g. reliability and maintenance concerns, to focusing on the process, e.g. improving methods and process models"*[1]

Future progress may well hinge on focusing on the People.

[1] From "Software Development: A Paradigm for the Future", Victor R. Basili, Proc. 13th Int'l Computer Software & Applications Conf. Orlando FL, Sep 89

PSP Status

The PSP was developed by Watts Humphrey

Several industrial organizations are now introducing PSP methods (DEC, HP, TI) with encouraging results

SEI is offering train the trainer courses

Several universities are teaching the PSP (CMU, U. of Mass., Howard U., Embry-Riddle U., McGill, and others)

The textbook "A Discipline for Software Engineering" and support diskette are available from Addison Wesley.

Questions

For more information or off-line discussion contact:

Daniel Roy
20 Forest Rd
Bradford Woods PA 15015
(412) 934 0943
dmr@sei.cmu.edu

Omit

Session 3: Certification

Applying Program Comprehension Techniques to Improve Software Inspections
Stan Rifkin, Master Systems Inc.

An Experiment to Assess the Cost-Benefits of Code Inspections in Large-Scale Software Development
Harvey Siy, University of Maryland

A Process Improvement Model for Software Verification and Validation
Jack Callahan, NASA Independent Software Verification and Validation Facility