

CITS5502 Software Processes

Personal Software Process

Unit coordinator: Arran Stewart

Sources

- W. S. Humphrey “Using a Defined and Measured Personal Software Process”, IEEE Software 1996
- Philip M. Johnson, Anne M. Disney, “The Personal Software Process: A Cautionary Case Study”, IEEE Software 1998

Personal processes

- How do we become better at software development, at an individual level?

Personal processes

- How do we become better at software development, at an individual level?

What are the skills involved in software development?

Possible skills

- Specific tools/languages (e.g. Java, git)
- Language paradigms (e.g. OO, functional, logical)
- Good practices in
 - specifying
 - design
 - coding
 - testing, static analysis
 - debugging
 - revision management
- Planning and estimating
- Problem-solving skills
- Working with large systems
- Working with constrained systems
- ... What else?

Practice

“Contrary to what you might believe, merely doing your job every day doesn’t qualify as real practice. Going to meetings isn’t practicing your people skills, and replying to mail isn’t practicing your typing. You have to set aside some time once in a while and do focused practice in order to get better at something.”¹

— Steve Yegge, programmer and blogger, formerly of Amazon and Google

¹<https://sites.google.com/site/steveyegge2/practicing-programming>

Practice cont'd

- Practice can still be bad or insufficient – no use practicing things un-reflectively, or that pose no challenge to you
- If you're not failing sometimes, are you actually practicing at the limit of your ability?

Typical training methods

- A typical “training method” for many companies is to send developers to a 3-day course on some technology, X – how effective is this?
- What factors might limit the effectiveness of this approach?

The PSP

The Personal Software Process (PSP) was created by Watts Humphrey to apply the principles of the Capability Maturity Model (CMM) to software development practices at the level of the individual developer

- Humphrey discovered that CMM did not work well over level 3 (particularly for small organization) without personal improvement.

PSP key concepts

- Bottom-up rather than top-down process improvement
 - i.e., improving the organization through individuals
- PSP particularly designed for small teams
- Establishment of personal process goals
 - Self measurement, analysis, adjustment of methods
- PSP learnt in 7 steps – through writing 10 programs
- Five framework activities
 - Planning, Design, Review, Development (Code & Test), PIR

- A number of studies attempting to measure the effectiveness of PSP
- One report² found improved performance in size estimation and effort estimation accuracy, product quality, process quality, and personal productivity, without any loss of overall productivity

²Will Hayes and James W. Over, *The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers*, Technical Report CMU/SEI-97-TR-001, Software Eng. Inst., Pittsburgh, 1997

Anecdotal reports:³

“[W]hen I started this course, I understood what we were supposed to do in good software engineering, but I never really did it. Now I understand the reasons behind these practices and the benefits of actually following a process instead of just jumping right into coding.”

³Philip M. Johnson, Anne M. Disney, “The Personal Software Process: A Cautionary Case Study”, IEEE Software 1998

General approach

- Collect data to determine a developer's initial personal process
- Generate and analyse a model of actual behaviour, that is accurate enough to support process improvement

PSP – A Series of Steps

Humphrey introduced the PSP concepts in a series of steps:

- Personal Measurement (PSP0). Engineers
 - learnt how to apply the PSP forms/scripts to their personal work
 - collected real data, benchmark against what they measured
 - recorded problems, issues, and ideas in Process Improvement Proposal (PIP) form to use later in improving their processes
- Personal Planning (PSP1). Engineers used the Proxy-based Estimating (PROBE) method to estimate
 - size
 - development timesfor new programs (based on their personal data)
 - They also did schedule and task planning

PSP – A Series of Steps

- Personal Quality (PSP2). In this step, defect management is introduced. Engineers
 - constructed and used checklists for design and code review
 - learnt why it's important to focus on quality from the start
 - learnt how to efficiently review their programs
- Scaling Up (PSP3). This is the final PSP step. Engineers were expected to be able to couple multiple PSP2 processes in a cyclic fashion to scale up to developing much larger modules

TABLE 1
PSP EXERCISES

Program Number	Brief Description
1A	Using a linked list, write a program to calculate the mean and standard deviation of a set of data.
2A	Write a program to count program LOC.
3A	Enhance program 2A to count total program LOC and LOC of functions or objects.
4A	Using a linked list, write a program to calculate the linear regression parameters (straight line fit).
5A	Write a program to perform a numerical integration.
6A	Enhance program 4A to calculate the linear regression parameters and the prediction interval.
7A	Using a linked list, write a program to calculate the correlation of two sets of data.
8A	Write a program to sort a linked list.
9A	Using a linked list, write a program to do a chi-squared test for a normal distribution.
10A	Using a linked list, write a program to calculate the three-parameter multiple regression parameters and the prediction interval.
1B	Write a program to store and retrieve numbers in a file.
2B	Enhance program 1B to modify records in a file.
3B	Enhance program 2B to handle common user errors.
4B	Enhance program 3B to handle further user error types.
5B	Enhance program 4B to handle arrays of real numbers.
6B	Enhance program 5B to calculate the linear regression parameters from a file.
7B	Enhance program 6B to calculate the linear regression parameters and the prediction interval.
8B	Enhance program 5B to sort a file.
9B	Write a program to do a chi-squared test for a normal distribution from data stored in a file.

Reports

- R1 LOC counting standard: Count logical LOC in the language you use to develop the PSP exercises.
- R2 Coding standard: Provide one logical LOC per physical LOC.
- R3 Defect analysis report: Analyze the defects for programs 1A through 3A.
- R4 Midterm analysis report of process improvement.
- R5 Final report of process and quality improvement and lessons learned.

PSP activities

Requires collection of data on lines of code. How many LOC are there?

```
/** This program adds two input numbers and prints out  
* their sum.  
*/  
#include <stdio.h>  
  
int main() {  
    int a, b;  
    printf("Please_enter_two_integers:_");  
    scanf("%d%d", &a, &b);  
    int result = a + b;  
    printf("The_total_is_%d\n", result);  
}
```

Halstead Complexity Measures

- Halstead complexity measures are software metrics introduced by Maurice Howard Halstead in 1977
- The metrics are designed to reflect the expression of algorithms in the program, independent of the programming language and execution platform
- In each module, count
 - n_1 – number of distinct operators
 - n_2 – number of distinct operands
- Let N_1 and N_2 be the total number of operators and operands

Halstead Complexity Measures (cont.)

- Several measures can be calculated from those numbers on the previous slide:
- Program vocabulary: $n = n_1 + n_2$
- Program length: $N = N_1 + N_2$
- Difficulty: $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$
- Volume: $V = N \times \log_2 n$
- Effort: $E = D \times V$
- Time required to program: $T = \frac{E}{18}$ seconds

PSP Methods – Gathering Data

- The PSP measures were defined with the Goal-QuestionMetric (GQM) paradigm.
- The data gathered in every process phase and summarized at project completion provide the engineers a family of process quality measures:
 - Size and time estimating error
 - Cost-performance index
 - Defects injected and removed per hour
 - Process yield
 - Appraisal and failure cost of quality
 - The appraisal to failure ratio

PSP Methods – Gathering Data (cont.)

- Controlling cost of quality: To manage process quality, the PSP used 3 cost-of-quality (COQ) measures:
 - Appraisal costs: development time spent in design and code reviews
 - Failure costs: time spent in compile and test, and
 - Prevention costs: time spent preventing defects before they occur (e.g., time spent on building prototypes and formal specification)

Another COQ measure was the appraisal to failure ratio, which is defined as Appraisal-cost / Failure-cost, denoted as A/FR.

PSP Methods – Estimating & Planning

- The PROBE method was used
 - Size proxies – number of objects and functions (other proxies are also possible)
- Planning accuracy was measured by the cost-performance index (i.e., the ratio of planned to actual development cost)
- Individual estimates generally had considerable error
- Overall, engineers' estimating ability improved considerably during the PSP course

PSP Methods – Managing Defects

- All defects were counted, including those found in
 - Compiling
 - Testing
 - Code inspection
- Defects were classified into 10 categories
- Defects injected must be fixed earlier, as they are more expensive to find and fix later on
- Engineers also kept track of the phases when the defects were injected and removed. The phases were: planning, design, design review, code, code review, compile, and test

PSP Methods – Managing Yield

- Yield is the principal PSP quality measure
- Total process yield is the %defects found and fix before the engineer started to compile and test the program
- The PSP in Humphrey's paper focused on defect detection and defect prevention because finding and fixing defects absorbs most of the development time and expense
- In PSP, engineers must review their code before the first compile
 - Evidence shows that the more defects were found in the compilation step, the more likely they would be present in the testing step (correlation = 0.71)

Discussion

- What advantages do you see in using any sort of principled PSP?

Advantages

- Easy for developers to only focus on urgent problems, and not long-term improvement of skills
- Can be helpful to commit to a program/plan and have organizational support
 - why do we sign up for gym programs?

- What disadvantages do you see in using a system like the PSP?

Possible downsides

- Inflexible?
- Time-consuming?

Time and paperwork requirements

- Developing a system using PSP 2.0 requires a developer to fill out “12 separate paper forms, including a project plan summary, time recording log, defect recording log, process improvement proposal, size estimation template, time estimation template, object categories worksheet, test report template, task planning template, schedule planning template, design checklist, and code checklist”.⁴
- Which gives 500 distinct values to be calculated or recorded

⁴Johnson & Disney, 1998

Empirical evidence discussion

- Based just on what we have discussed in class, do you have any critiques of the studies examining the effectiveness of the PSP?

Studies from the Hawthorne electrical works

- Henry A. Landsberger analyzed earlier experiments from 1924–32 at the Hawthorne Works (a Western Electric factory outside Chicago).
- The Hawthorne Works commissioned a study to see if workers would be more productive in higher or lower levels of light.
- Various changes made . . .
 - giving different lengths of breaks
 - providing food during the breaks
 - shortening the day, and returning it to normal

⁵But note there are criticisms of the interpretation of the Hawthorne effect, too

Studies from the Hawthorne electrical works

- Henry A. Landsberger analyzed earlier experiments from 1924–32 at the Hawthorne Works (a Western Electric factory outside Chicago).
- The Hawthorne Works commissioned a study to see if workers would be more productive in higher or lower levels of light.
- Various changes made . . .
 - giving different lengths of breaks
 - providing food during the breaks
 - shortening the day, and returning it to normal
- *Changing* a variable usually increased productivity, even if this was just a change back to the original value.⁵


⁵But note there are criticisms of the interpretation of the Hawthorne effect, too

Factors to consider

- Could effects derive just from the Hawthorne effect? (Unlikely)
- The PSP lays out very specific steps and exercises – but which ones cause the effect? Must someone use *exactly* this process, exactly these forms, exactly these exercises? (Seems unlikely)
 - Could the same effect be got, with less paperwork?
- What steps are taken to remove bias? Can you “blind” experimenters?
- Are you comparing against the best known current practices?
- Are any effects due to your sample population?

From Steve Yegge's "practicing programming" blog post:⁶

- Practice Drill #1: Write your resume. List all your relevant skills, then note the ones that will still be needed in 100 years. Give yourself a 1-10 rating in each skill.
- Practice Drill #2: Make a list of programmers who you admire. Try to include some you work with, since you'll be borrowing them for some drills. Make one or two notes about things they seem to do well — things you wish you were better at.
- Practice Drill #3: Go to Wikipedia's entry for computer science, scroll down to the "Prominent pioneers in computer science" section, pick a person from the list, and read about them. Follow any links from there that you think look interesting.

⁶<https://sites.google.com/site/steveyegge2/practicing-programming> 

Practicing

- Practice Drill #4: Read through someone else's code for 20 minutes. For this drill, alternate between reading great code and reading bad code; they're both instructive. If you're not sure of the difference, ask a programmer you respect to show you examples of each. Show the code you read to someone else, and see what they think of it.
- Practice Drill #5: Make a list of your 10 favorite programming tools: the ones you feel you use the most, the ones you almost couldn't live without. Spend an hour reading the docs for one of the tools in your list, chosen at random. In that hour, try learn some new feature of the tool that you weren't aware of, or figure out some new way to use the tool.

Practicing

- Practice Drill #6: Pick something you're good at that has nothing to do with programming. Think about how the professionals or great masters of that discipline do their practice. What can you learn from them that you can apply to programming?
- Practice Drill #7: Get a pile of resumes and a group of reviewers together in a room for an hour. Make sure each resume is looked at by at least 3 reviewers, who write their initials and a score (1-3). Discuss any resumes that had a wide discrepancy in scoring.
- Practice Drill #11: Find a buddy for trading practice questions. Ask each other programming questions, alternating weeks. Spend 10 or 15 minutes working on the problem, and 10 or 15 minutes discussing it (finished or not.)

Other non-SEI PSP techniques

- Reading. Many engineers and software developers *rarely read* any texts on software development.
 - Is reading details of languages/algorithms online sufficient?
- Monitoring. How do you know if you're getting any better?
- Practicing.
- Attending conferences and workshops. Make sure you're getting something out of them. It needn't always be what's advertised.