Personal Software Process

Idaho State University | Computer Science

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

ROAR

Outcomes

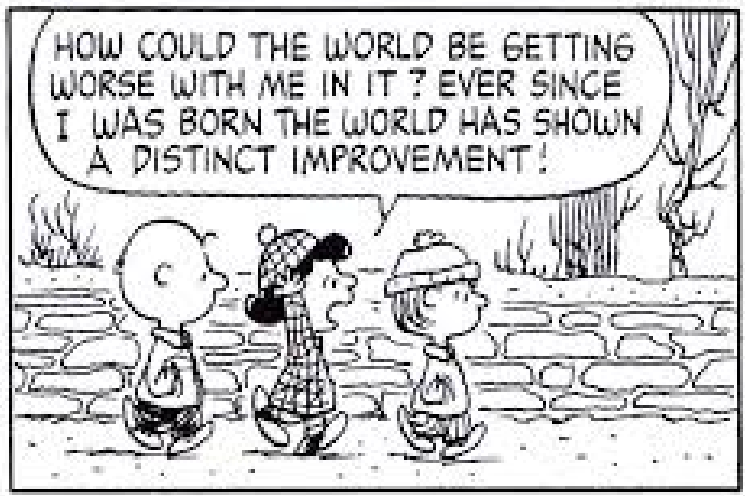
After today's lecture you will be able to:

- Understand the basic notions of the PSP
- Understand the importance of the PSP
- Be capable of applying the concepts of PSP to your daily practice

Inspiration

"One principle problem of educating software engineers is that they will not use a new method until they believe it works and, more importantly, that they will not believe the method will work until they see it for themselves."
– Humphrey, W.S.

PSP


HOW COULD THE WORLD BE GETTING WORSE WITH ME IN IT ? EVER SINCE I WAS BORN THE WORLD HAS SHOWN A DISTINCT IMPROVEMENT !

The Personal Software Process (PSP)

- The **software process** is about making software engineering groups/teams work to the best of their abilities
- The **personal software process** is about making individual engineers work to the best of their abilities
- Central to both is feedback —
  - through analysis of practical application of the process, the process should be changed for the better
- Software engineers should accept responsibility for the quality of their work
- Software engineers can do this only if they have a way of evaluating quality and improving quality (through experience)
- The software process improves individual engineers to some extent, but it is possible for a project to succeed even when an individual participant has not!
- The PSP is individual oriented: it is possible for an individual to succeed within a project that fails.

The Personal Software Process (PSP)

PSP is a structured software development process that is intended to help software engineers understand and improve their performance, by using a “disciplined, data-driven procedure”:

- **Improve their estimating and planning skills.**
- Make commitments they can keep.
- Manage the quality of their projects.
- Reduce the number of defects in their work.

The PSP was created by Watts Humphrey to apply the underlying principles of the SEI’s Capability Maturity Model (CMM) to the software development practices of a single developer.

Criticisms of the PSP

The PSP is not universally accepted:

- some think it is a good idea in theory but not in practice
- some think that it is not flexible enough
- some think that it is too time consuming
- some think it should be up to individuals to find their own way of working

In fact, the PSP is only a framework of common sense ideas and suggestions which engineers are encouraged to think about as they learn from experience

The PSP fails, IMHO, when it is used by the team to criticize individuals. This risk occurs when the PSP is used, rightly, to give feedback to the team-wide process.

… make up your own minds …

Overview of PSP

There are four main areas to examine:

- assumptions
- process stages
- measures; basic and derived
- results: training and industry

In the case study after this set of lectures you will be asked to run your own PSP while developing a small piece of code.

This alone is not enough to let you judge the merits of PSP

It is enough to give you an idea of how to carry out the process

PSP Assumptions

- Software engineers currently learn software development by developing toy programs
- They develop their own process since process is not taught in introductory classes
- These toy processes do not provide a suitable foundation for large-scale software development
- To use effective methods consistently, engineers must believe that they are effective
- To believe that they are effective, they must use them
- To teach effective system processes we need to start with large system practices, select those that are suitable for individuals and introduce them incrementally

PSP Process Stages

- PSP 0 Current Process, Basic Measures
  - PSP 0.1 coding standard, process improvement proposal, size measurement
  - Leads to: **Personal Measurement**
- PSP 1 Size estimating, test, report
  - PSP 1.1. task planning, schedule planning
  - Leads to: **Personal Planning**
- PSP 2 Code reviews, design reviews
  - PSP 2.1 design templates
  - Leads to: **Personal Quality**
- PSP 3 Cyclic Development
  - Leads to: **Scaling Up**

PSP 0: Personal Measurement

- Engineers gather data on the time they spend by phase and the defects they find
- Generates real, personal data and provides the base benchmark for measuring progress
- 3 Phases: planning, development and postmortem
- PSP 0 adds a coding standard, size measurement and a process improvement proposal

PSP 1: Personal Planning

- This step must introduce some method for estimating sizes and development times for new programs based on personal data
- The methods employed are usually (should be) based on linear regression with prediction intervals to indicate size and estimate quality
- PSP1.1 adds schedule and task planning

PSP 2: Personal Quality

- This step introduces defect management
- Using data from PSP exercises, engineers construct and use checklists for design and code review
- From their own data, they see how checklists help personal reviews
- PSP 2.1 adds design specification and analysis techniques along with defect prevention, process analyses and process benchmarks

PSP 3: Scaling Up

- The final step shows how engineers can couple multiple processes in a cyclic fashion to scale up to developing systems with many thousands of lines of code (LOC)
- It uses an iterative enhancement approach
- A team software process should be developed as the next step for systems larger than 10K LOC

PSP Programming Exercises

The following exercises are widely used and accepted as providing a good case set on which to start developing a PSP:

- calculate mean and standard deviation of numbers in a linked list
- count LOC in a source program
  - enhance to count total and function LOC
- calculate linear regression parameters
- perform numerical integration
  - enhance to calculate prediction interval
- calculate correlation of 2 lists
- chi-squared tests for normal distribution
- calculate multiple regression parameters

PSP Basic Measures

- **Development Time:** measured in minutes (!) using a time recording log designed to account for interruptions
- **Defects:** any change to the design or code to get the program to compile or test correctly; recorded in a defect recording log
- **Size:** lines of code, used primarily for estimating development time; new, modified and re-used code is distinguished.

PSP Derived Measures

- Estimating accuracy — time and size
- Test defects/KLOC
- Compile defects/KLOC
- yield: % of defects injected before 1st compile that are removed before 1st compile
- appraisal time — time in review
- failure time — time in compile and test
- cost of quality – appraisal time + failure time
- appraisal/failure ratio

PSP Quality Strategy

- Defects are basic quality measure
- Engineers should
  - remove them
  - determine their cause (type)
  - learn to prevent them
- PSP uses private review with the goal of finding all defects before 1st compile and test

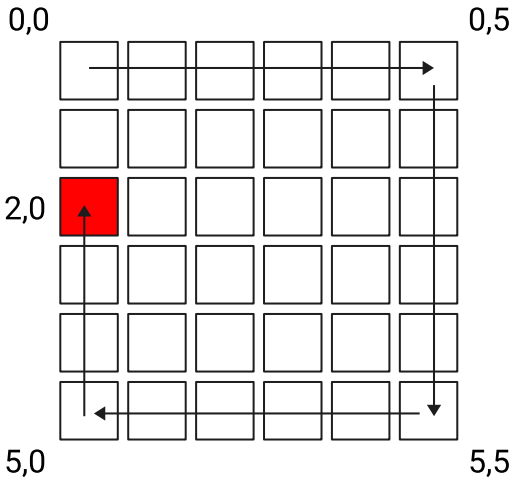
PSP Training Data

- Each programming assignment results in approx. 70 pieces of data being collected by each engineer
- It is collected and collated by instructors to provide feedback during training
- Thre is a well cited study based on 23 PSP classes consisting of 298 engineers, over 300,000 LOC during > 15,000 hours, about 22,000 defects were found and removed
- Each analysis is based on at least 170 cases where complete data was available

PSP Statistical Analysis

- Large individual differences are expected when measuring software engineering performance
- Consequently, rather than studying changes in group averages, the study focuses on the average change in engineers
- The repeated measures of variance method analyses the differences across multiple trials to uncover trends

PSP Sample Exercise (spiral walker)



- You are to implement a function, f, in Java that takes as input the:
  - size of a square grid
- Calculates the x,y coordinates of the robot after it has walked half way around the grid following a spiral walk, starting at 0,0 and moving clockwise.
- In the example the function calculates
  - f(6) = (2,0)

PSP Sample Exercise (spiral walker)

- Estimate: time of development, LOC
- Note/Count/Measure (for every version, i.e., every time you compile and run):
  - Compile defects - syntax/semantic errors
  - Test defects - code is wrong, design is wrong, test is wrong?
  - LOC - implementation/tests
  - Comments/Code ratio
  - Time spent on each task between versions - testing, coding, designing, commenting
  - The reason for the compile/run - expected outcome versus actual outcome

Are there any questions?