

# CSC 7003 : Basics of Software Engineering

**J Paul Gibson, A207**

`paul.gibson@int-edu.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/>

## **Personal Software Process**

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/L2-CSC7003/PSP.pdf>

# 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 Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer.

- "Using a defined and measured Personal Software Process" by Watts S. Humphrey, published in IEEE Software, May 1996, pages 77-88.

# Criticism 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 criticise individuals. This risk occurs when the PSP is used, rightly, to give feedback to the team-wide process.

... make up your own minds ...

A good criticism is found in: Philip M. Johnson, Anne M. Disney, "*The Personal Software Process: A Cautionary Case Study*," IEEE Software, pp. 85-88, November/December, 1998

## 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

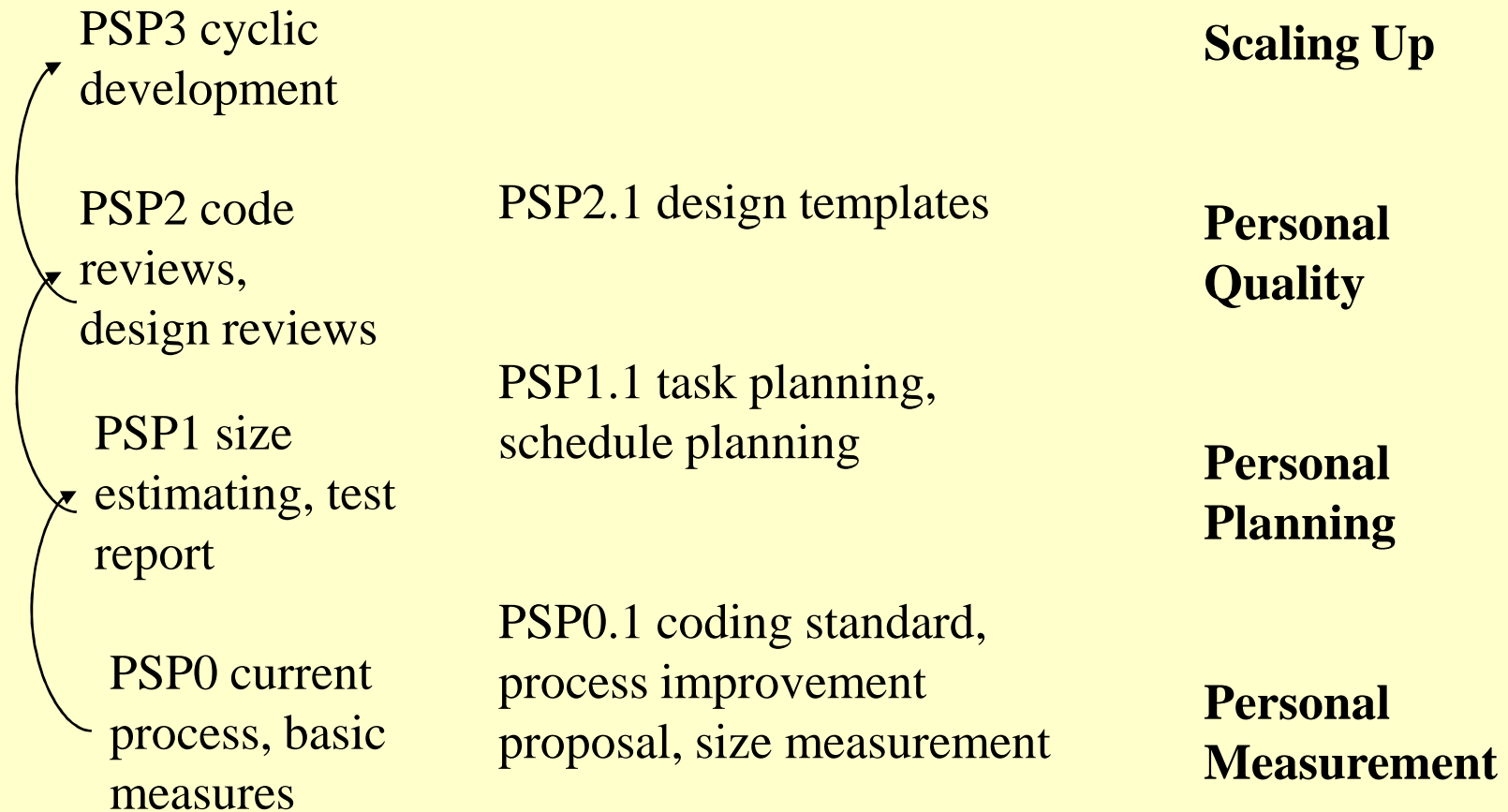
I took these from standard texts:

- *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*

There is, IMHO, a degree of truth to each of these

# PSP Process Stages

These are similar to the CMM for development processes.



# PSP0: 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

PSP0 adds a coding standard, size measurement and a process improvement proposal

# PSP1: 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

## **PSP2: 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

PSP2.1 adds design specification and analysis techniques along with defect prevention, process analyses and process benchmarks

## PSP3: Scaling Up

The final step shows how engineers can couple multiple 3/PSP2.1 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

There 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 Encouraging Results ---

The following claims for CSC7003/PSP have been made and *accepted* (?)

---

Size estimation usually improves by a factor of  $>2.5$

Defect density reduces at each level

- median reduction in compile defects is 3.7
- median reduction in test defects is 2.5
- median improvement in number of defects removed before compile  $> 50\%$

Productivity (measured in LOC/hour):

- between CSC7003/PSP0 and CSC7003/PSP1 there is a decrease (1 LOC/hour)
- between CSC7003/PSP1 and CSC7003/PSP2 there is an increase (2 LOC/hour)

**Question:** what are your feelings about CSC7003/PSP?

## **PSP: Sample Exercise (longest substring palindrome)**

You are to develop a Java function that will find the longest palindrome in any given string of characters.

A palindrome is defined as being the same forwards as backwards, eg « ABBA » is a palindrome but « ABBAB » is not

For example:

Input GCABBACEEEEF should give the result CABBAC

Where there is no unique longest palindrome then you are to return the list of all such palindromes.

For example:

Input CABBADEEEEF should give the result ABBA, EEEE

## **PSP: Sample Exercise (longest substring palindrome)**

**Estimate: time of development, LOC**

**Count/Measure (for every version):**

- **Compile defects**
- **Test defects**
- **LOC – implementation/tests**
- **Comments/Code ratio**
- **Number of versions**

**NOTE: After all of you have solved the problem we will analyze the PSP ‘results’**