

Scaling Up the Personal Software Process

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 1

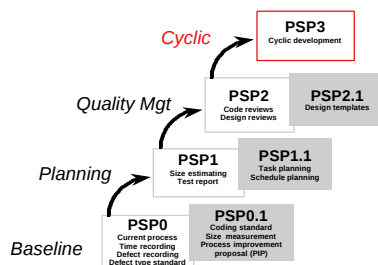
Outline

- Review of PSP Levels
- Overview
- Abstractions
- Stages of Product Size
- Developing Large-scale Programs
- A Potential Problem with Abstractions
- The Development Strategy
- PSP3
- Homework #7 - Part 1

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 2

Review of PSP Levels (Humphrey, 1995, p. 11)



AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 3

Overview (cf. Humphrey, 1995, p. 353-354)

- The size of a similar software product increases an order of magnitude every 5-10 years.
- Ex: HP Laserjet software
 - LJ - 25,000 LOC
 - LJ-II - 200,000 LOC
 - LJ-III - 1,000,000 LOC
- Therefore, your software development process needs to be able to scale up over time.
- In this section we discuss problems, principles, and strategies associated with developing large-scale systems. The PSP3 is one example of how to do this.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 4

Abstractions (cf. Humphrey, 1995, p. 354-356)

- Physical scientists use abstractions and laws to help abstract away the confusing details.
- Computer scientists cannot abstract away details, because the system will most likely become unusable.
- However, we are free to build and use whatever abstractions we wish. We just need to make these abstractions consistent and complete.
- Our work is intellectual, and has three components:
 - **Memory:** People can usually only remember 7 +/- 2 "chunks", but patterns can enhance the amount of detail we can keep track of.
 - **Skills:** As we gain skills and experience, the number of "patterns" with which we are familiar grows, and thus so does our development ability.
 - **Methods:** By breaking down large processes for large projects into smaller sub-processes we can manage large development efforts.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 5

Stages of Product Size (cf. Humphrey, 1995, p. 356-361)

Stage	Description
0	- Σ Very small program elements. - Σ Written by programmers alone.
1	- Σ Small programs or modules. - Σ Designed, implemented, tested by programmers alone.
2	- Σ Larger programs or components. - Σ Typically developed by teams who develop & integrate multiple Stage-1 modules into larger Stage-2 components.
3	- Σ Very large projects. - Σ Involve multiple teams controlled & directed by a central project management.
4	- Σ Massive multisystems. - Σ Involve many autonomous or loosely federated projects.

- Within each range a given process is likely applicable to many projects.
- When you cross a scalability boundary you will need new process features.
- Your boundaries are dependent on and change with your skills and abilities, thus your boundaries change over time.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 6

Stage-0: Simple Routines

- **Smallest building blocks:**
 - loops, if-then-else, ...
- **Experienced programmers do not design these constructs - that would be like designing how to add a string of numbers...**

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 7

Stage-1: The Program or Module

- 10's - several 100's LOC
- Design in your head, type in, compile.
- Beginning programming classes:
 - 300 LOC
 - written from scratch
 - in a "dead" language
 - "clear" boxes
- Properties:
 - Not scaleable - can't continue to use intuitive methods to build large programs
 - By using purely intuitive methods, programmers don't develop scaleable methods.
 - Programmers may attempt to use these (familiar) methods on large-scale systems, unsuccessfully.
- Moving from 1->2
 - Interact with other developers and get ideas from them for the new and unfamiliar things with which you must now deal. *cf. Fig 11.1, p. 358*

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 8

Stage-2: The Component

- Entire programs are abstractions.
- Visualize interconnecting Stage-1 modules.
- Processes beyond their capacity at Stage-2 have two symptoms:
 - Inadequate design
 - Overlooked detail
- Problems:
 - Many details
 - Assumption of correctly working interacting modules
- Here you need good quality control and disciplined practices, and must work effectively in teams.
- Moving 2->3
 - Must master larger-sized programs
 - Must have and follow system standards, especially for early defect prevention & removal.
 - Must practice defensive programming and design for testability.
 - Team relationships must become more formalized, and must be supported by formal team processes.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 9

Stage-3: The System

- Work with large multi-component systems.
- Understand the external interfaces of these components, but not their inner workings.
- Problems:
 - Hiding functional complexity from users (so they are not overwhelmed with the multitude of capabilities).
 - Maintaining component quality: integration is difficult if not impossible with low quality components.
- Your PSP could totally change, or become totally focused on a small part of the overall process.
- Moving 3->4
 - Reduce centralized control, because:
 - No one could possibly track all the activities.
 - No one could understand all the components.
 - Too many communication paths would be necessary.
 - Data to central control would be late & incomplete, and would thus lead to poor decision-making.
 - Centralized control de-motivates the people at the bottom, who need to take effective action on their own.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 10

Stage-4: The Multisystem

- While system-wide standards, communication methods, and processes are required to manage multi-systems, the subsystems are developed under quite independent teams, with independent requirements.
- Requires:
 - Extraordinary quality.
 - Security, access authorization, audit trails...
 - Know and follow system standards precisely.
 - Thus developers must be highly disciplined.

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 11

Developing Large-Scale Programs (cf. Humphrey, 1995, p. 361-364)

- Approaches to developing large-scale systems:
 - Use your or someone else's prior process
 - You have built a similar product
 - Start & explore - Boehm's spiral model
 - You know how to start but not how to complete it
 - Prototype / throw away
 - You don't even know how to start
 - It is unlikely you'll build a system understanding by following an iterative incremental process
- Large-scale development is disintegration (design) and reintegration (integration) - your process must support this.
- Large systems evolve by enhancement and accretion of smaller systems
 - interfaces adapt between the smaller systems
 - there must be structured methods for understanding and controlling changes, and for capturing and disseminating knowledge

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 12

Scaleable Systems (cf. Humphrey, 1995, p. 363)

- A SW system is scaleable if:
 - it can be disintegrated into smaller components
 - the smaller components can be developed
 - the system can be reintegrated (without modifying the components during integration)
 - it has an essence - conceptual integrity

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 13

A Potential Problem with Abstractions (cf. Humphrey, 1995, p. 364-365)

- Just breaking down a system into fewer smaller pieces does not automatically solve the scalability problem
 - Ex: 1,000,000 LOC
 - 500 5LOC parts created
 - 200,000 unfamiliar parts still must be dealt with
- In order to have useful scalability, the system must be subdivided, but the parts must at the same capture significant system functionality

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 14

The Development Strategy (cf. Humphrey, 1995, p. 365-368)

- A good development strategy:
 - Naturally matches the system's structure
 - Exposes key risks as early as possible
- There are many strategies, none of which are the single best strategy - each has advantages and disadvantages.
- You must choose a strategy that best fits your project.
- Several strategies:
 - Progressive ("pipeline")
 - System processes information in a sequential manner
 - Functional Enhancement
 - Kernel + enhancements, see working system earliest
 - Fast-Path Enhancement
 - Demonstrate key timing/system problems as early as possible
 - Dummy
 - Top-down, layered, good for kernel of enhancement approaches

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 15

PSP3 (cf. Humphrey, 1995, p. 368-371)

- Principal role
 - an example of a foundation process for large-scale SW development
- Therefore it must handle increased complexity and be able to relate to team processes
- cf. Fig 11.3, p. 369, for overview

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 16

The Overall PSP3 Approach

- Plan conceptual design, estimate size, plan development work
- High level design subdivides work
 - These will define activities for subsequent cycles
 - 100-300 LOC (new & changed) per cycle
- For each cycle
 - establish spec's for current cycle
 - follow regular development process for the current sub-system
 - be especially attentive to quality (thorough reviews, defect prevention, removal) since subsequent cycles will use this code
- Develop tests and perform reviews
 - test development may find as many defects as testing does
 - revise tests / reviews based on information from the other
- Reassess & recycle
 - Determine your status and reevaluate your plan
 - Check data against plan / schedule and update if necessary

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 17

Homework #7 - Part 1

(cf. Humphrey, 1995, p. 353-354)

- See "Homework Assignments" listing and textbook instructions

AU INSY 560, Singapore 1997, Dan Turk

Humphrey Ch. 11 - slide 18