INF5180: Software Product- and Process Improvement in Systems Development

Part 04:

Problem Solving and Improvement – Individually

UNIVERSITETET I OSLO

Dr. Dietmar Pfahl

email: dietmarp@ifi.uio.no

Spring 2010

Software Engineering is Problem Solving

UNIVERSITETET I OSLO

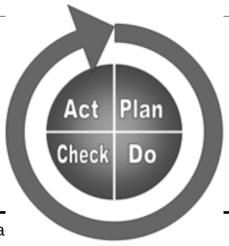
”Can you please solve this problem for me?”

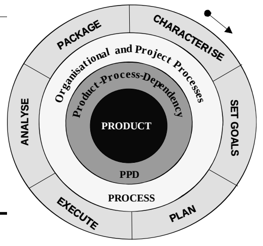
- ”OK, I’ve solved similar problems before. Can you describe a bit closer what you wish to get?”

  .....
- ”I’ve now started to solve the problem. Do you want <A> or <B>?”

  .....
- ”Look, here is the solution! Isn’t it nice? Does it satisfy your need?”

A general process for solving problems:
1. Understand the problem
2. Design and realize a solution
3. Verify & validate the solution





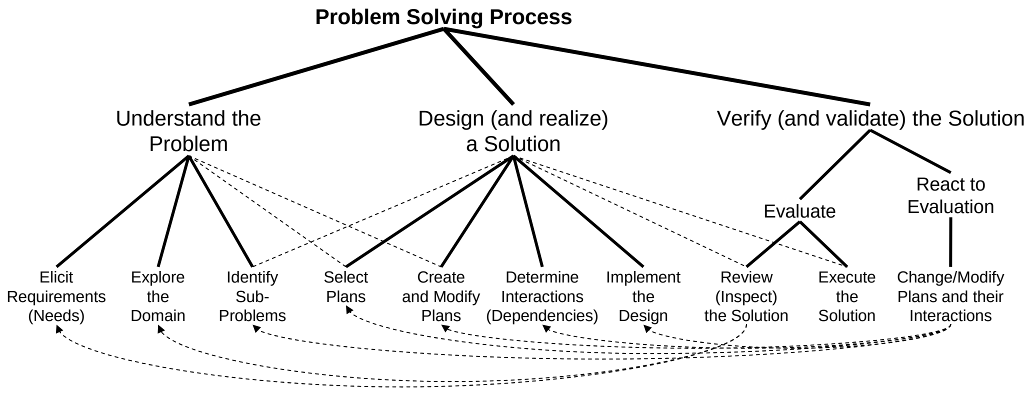
Problem Solving Strategy – Divide and Conquer

- A problem can always be split into sub-problems which can further be split etc...
- Splitting-up increases the level of detail which, in turn,
  - increases accuracy
  - slows down progress

- Process for “divide & conquer”:
  1. Define the problem
  2. Split-up the problem into sub-problems which can be solved, and

     repeat this until all sub-problems can be solved
  3. Integrate sub-solutions so that it solves the original problem

Universal Procedure for Problem Solving [Hohmann – Ch. 1.1.]


Problem Solving Process
Understand the Problem
Design (and realize) a Solution
Verify (and validate) the Solution
Evaluate
React to Evaluation
Elicit Requirements (Needs)
Explore the Domain
Identify Sub-Problems
Select Plans
Create and Modify Plans
Determine Interactions (Dependencies)
Implement the Design
Review (Inspect) the Solution
Execute the Solution
Change/Modify Plans and their Interactions

UNIVERSITETET I OSLO

Problem Solving – Methods

- Method = “a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well-defined notation.” (Booch)
- Notes:
  - It is nonsense to say that one method is (always) better than another
    - NB: The appropriateness of a method is problem, situation, and person dependent.
  - Within a project (or organization) only <u>one</u> (most appropriate) method should be chosen.
    - This is sometimes not easy to achieve.
    - The worst thing is to let choose everybody their own method. (**Question**: Why?)

UNIVERSITETET I OSLO

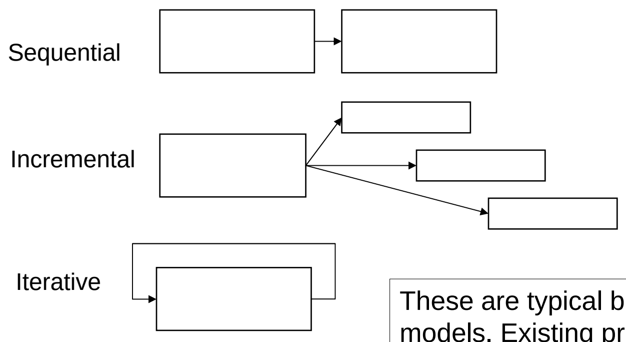
Problem Solving – Methods

- Describe systematic procedures to make better systems by providing structures that:
  - “automate” parts of the problem solving process
    - e.g. standardized refinement into sub-problems via “architectural styles” and design patterns
  - facilitate collaboration during the problem solving
    - e.g., by dividing the development into phases, and by using interface descriptions and coding standards
  - counteract typical “weaknesses” in humans
    - e.g., it is tempting to directly jump to the problem solution (the code) before the problem is understood (the analysis)
  - simplify reuse of experience
    - e.g., through that everyone uses the same development models and coding standards, and perhaps pair-programming and formal inspections

Problem Solving – Mental Models (Plans)

- What Hohmann calls ***Plan*** is a stereotype solution to a problem. It is also a **private** solution that only exists in the head of a person who has solved similar problems before (i.e., it is a *Mental Model*).
- A pattern is an **externalized** and **generalized** plan ($\rightarrow$ conceptual model)
  - *Design Patterns* are just this: experts used time and effort to describe solutions to design problems that you repeatedly come across in software design. For a *pattern* to be applicable to many (similar) problems, it must be generic.
- Problem solving can be regarded as searching, selecting, modifying, using and reusing of (**mental) models** for different purposes.
  - Experience and the ability to solve problems is largely determined by the amount (and quality) of such (mental) models.

Software Engineering Process Models



These are typical building blocks of process models. Existing process models are combinations of these.

Software Engineering (Process) Models

How well do (process) models support our problem solving approach?

- Sequence: some (sub-)processes need outcomes from other (sub-)processes as inputs
- Increment: some (sub)-processes can be conducted in parallel; similarly, outcomes (products) might be decomposed and processed independently from each other
- Iteration: sometimes (sub-)processes need to be repeated (e.g., in order to correct/adjust outcomes)
- Combinations: usually, different situations (task size and complexity, available resources, etc.) require varying degrees of combinations:
  - Is it always possible to capture all requirements of a product in the very beginning?
  - Is it not wiser (for large systems) to start working on the high-priority requirements and then learn during the development process and iteratively feed in additional requirements?
  - Can (throw-away) prototypes be useful for eliciting requirements (and for exploring new designs or technologies)?
  - …

Other Software Engineering Models

Product Models & Structures

- Architectural Styles
- Design Patterns
- Frameworks
- UML Models (Use case, Statechart, Sequence diagram, Class diagram, etc.)
- Communication Protocols
- PPD-Model (→ PROFES method)
- …

Product-Process-Dependency Model

- PPD Example taken from: D. Hamann, D. Pfahl, J. Järvinen, R. van Solingen (1999) “The Role of GQM in the PROFES Improvement Methodology”, in: Proceedings of 3rd Conference on Quality Engineering in Software Technology (CONQUEST 1999), pp. 64-79.

PPD Model 1.3.1		
	Technology Application Goal	
Technology	**Software Inspections**	
Product Quality	**Reliability**	
Process	**ENG.3 Software Requirements Analysis**	
	Technology Application Context	
CF.1	Experience of inspection team	low average ***high***
CF.2	Management commitment	low ***high***
CF.3	Overall time pressure	***low average*** high
CF.4	Module affected by new hardware	old_hw ***new_hw***
CF.5	Module developed externally	internally ***externally***

CF = Context Factor

Problem Solving – Mental Models (Plans)

- What Hohmann calls ***Plan*** is a stereotype solution to a problem. It is also a **private** solution that only exists in the head of a person who has solved similar problems before.
- A pattern is an **externalized** and **generalized** plan (→ conceptual model)
  - *Design Patterns* are just this: experts used time and effort to describe solutions to design problems that you repeatedly come across in software design. For a *pattern* to be applicable to many (similar) problems, it must be generic.
- Problem solving can be regarded as searching, selecting, modifying, using and reusing of (**mental) models** for different purposes.
  - Experience and the ability to solve problems is largely determined by the amount (and quality) of such mental models.

$\xrightarrow{}$ What is an Expert?

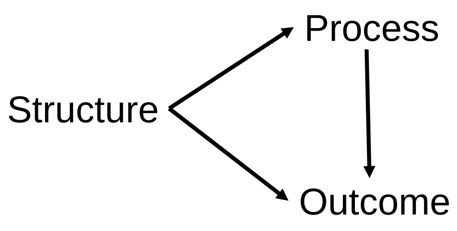
UNIVERSITETET I OSLO

What is an Expert?

- Experts have domain knowledge and experience (payroll systems, transportation systems, medical devices, communication systems, web-design, e-commerce etc..)
- Experts have method knowledge that is relevant
- Experts have technology knowledge and experience (Java, Unix, Web-services etc...)
- Experts have *bigger arsenal of mental models* (“*cognitive library of plans*”) related to domains and technologies.
- Experts are better in “chunking” (handling complex information in bigger entities).
  - They can, in other words, work on higher abstraction level.
  - Sub-problems become *details* with implicitly assumed solutions.

UNIVERSITETET I OSLO

Structure – What is it?



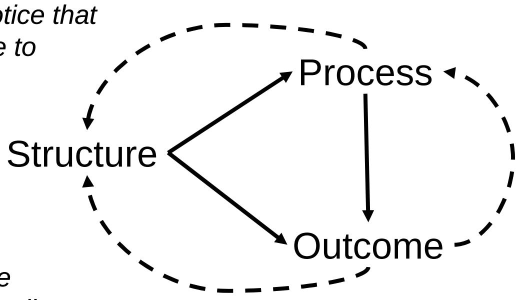
- Structure *defines* the form and content of outcomes

  and

  supports the processes we use to create them

Feedback

As (planned) Processes are carried out, one might notice that adjustments have to be made



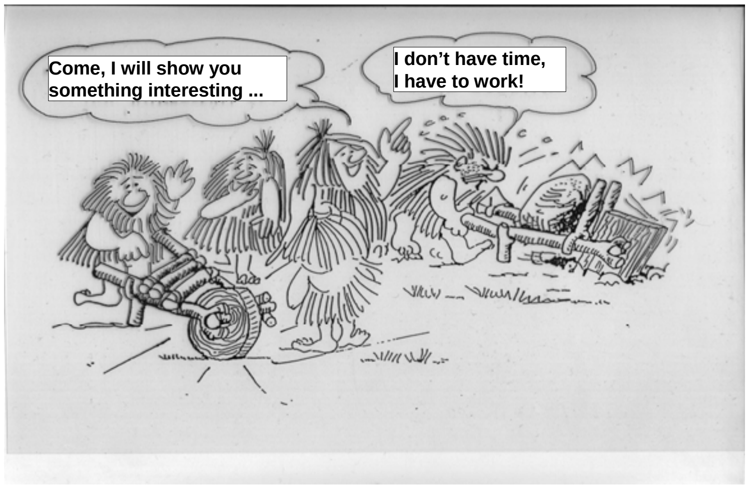
Processes adapt to (previously produced) Outcomes - partly due to convenience, partly to optimize

As (planned) Outcomes are Implemented, one might notice that adjustments have to be made

Structure – How much and what?

- How much and what structure is needed to achieve optimal problem solving (i.e., system/software development)?
- Issues:
  - Depends on problem and person(s):
    - Bigger and more complex problems typically need more structure
    - Experienced people need other types of structure than inexperienced
  - The more structure, the more standardization → standardization facilitates reuse of experience. (→ “design pattern”).

A typical situation?


Come, I will show you something interesting ...
I don’t have time, I have to work!

Structure – How is it introduced?

- Direct supervision and monitoring (by one who knows the processes and products)
- Using prescriptive standards of the processes (process handbooks)
- Using prescriptive standards of the product (product specifications)
- Standardizing skills (→ training)
- Mutual adoption, e.g.
  - Structures that facilitate collaboration are introduced and agreed upon ad-hoc
  - Continuing interchange with the customer defines the product structure
- …

Expanding the SPO-Framework

- Since the key element in software/systems development are people, the SPO-framework must be expanded to include several ”softer” factors that govern human behavior:
  - Values
  - Personality
  - Goals

Values – What are they?

- In the SPO-context values are:
  - Concepts or principles that are
    - deemed worthy or important for concrete choices (e.g., of methods)
    - not supported by (rational) arguments or perhaps not even articulated
  - What takes over when rational decisions cannot be made (e.g., two methods seem to be equally good)
- None of the descriptions above are precise or especially complete. It should still not be difficult to agree that values (with an intuitive understanding) are important for process improvement

Values – Examples

- If managers and developers in an organization have a consensus-culture (the “no one should be forced but convinced through argumentation”-value $\rightarrow$ Japan) it affects the decision processes.
  - Sometimes this culture will make a good improvement proposal fail because it wasn’t possible to get everyone to agree.
- Often “decision-happy” managers (the “leaders should make quick decisions”-value) starts too many improvement activities at once.

Personality

- “A personality is a complex set of relatively stable behavioral and emotional characteristics that can be used to uniquely identify a person.” (Hohmann)
- “Personality represents those characteristics of the person that account for consistent patterns of behavior.” (Pervin, “Personality”).

Elements:

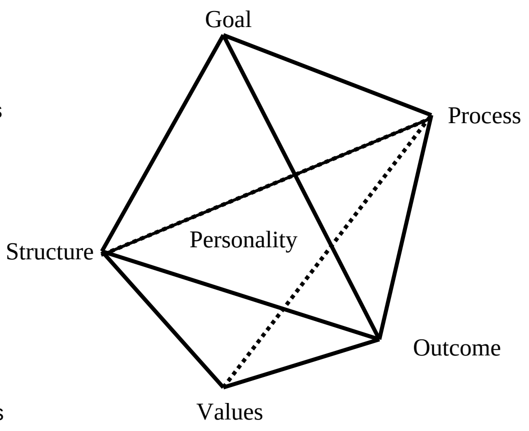
- Cognitive style
- Mental set
- Self-efficacy
- Assertive/Passive
- Tolerance of anxiety
- Tolerance for ambiguity
- etc…

Goals

- Have long-term influence on our behaviour
- The goals of those involved in process improvement activities are important for several reasons:
  - Process changes should be streamlined to help people achieve their goals (or at least not impede the achievement of their goals)
  - An organization works best when there is “a match” between personal objectives and organizational goals
    - It is too narrow to look at salary as the only (and possibly not the most important either) goal for a developer.
    - Equally important: recognition, professional pride, team experience, etc.

Framework Summary

- Structure – Process – Outcome:
  - Focus on control, support and standardized problem solving methods (sw/system development methods).
  - It is these elements (often not clearly separated from each other) that system development methods focus on.
- Values – Personality – Goals:
  - Represent to the “human side” of SPO.
  - These elements are rarely (explicitly) considered in sw/system development methods and little research about their effects on sw/system development has been conducted.



Conclusions

People are not equally comfortable with certain degrees of structure.

- New and innovative organizations, attract a special type of people (creative innovators who thrive in little structure). These people may have adaptation problems in a bigger and older organization with greater need for structure.
  - For example, a company founder is often not the best choice to lead the company after it has grown big (often, however, the founder himself/herself has difficulties to realize this).
- Bigger, older IT-organizations (typically government administration, bank/insurance, defense sector etc..) are often more plan-driven and documentation-heavy and want to attract confidence-seeking persons who thrive best with predictability.

Conclusions (cont'd)

Not everybody is like you

- It is easy for us to assume that others like the same and react equally as we do. For example, if a process improver prefers a high degree of structure he/she could easily assume that others also do, and react irrationally ("they work against me") if resistance is big.
- We like those who are like us, and devalue those who are different. As a consequence, we have a tendency to collaborate with those who have similar preferences regarding structure than we have.

Conclusions (cont'd)

- Groups that work on process improvement should be composed of persons with different personalities.
- It is not unreasonable to assume that a successful process improvement team or system development team needs:
  - Renewers/innovators (specially important in the start phase)
  - Researchers/launchers (specially important in the start phase)
  - Surveyors/developers (specially important in the start phase)
  - Pursuers/organizer (specially important in introduction and the follow-up phase)
  - Completers/producers (specially important in the introduction phase)
  - Informers/advisers (specially important in the introduction phase and the follow-up phase)
  - Supporter/maintainer (specially important in the introduction phase and the follow-up phase)
  - Controller (specially important in the follow-up phase)
- The big problems arise if important roles are not covered. For example, if there are no completers or controllers.

Conclusions (cont'd)

- Structuring of processes (process improvement) should get a balance between:
  - supporting preferred working manners
  - reducing the damaging effects of preferred working manners

Exercise

- Imagine an organization that implements web-solutions.
- The organization was started by two students at IfI and has in three years grown from two to forty employees.
- The founders have (with little help) realized that others ought to manage the organization and hire Petter who was a middle level manager in the IT-department of a bigger Norwegian bank.
- Petter sees immediately the need to introduce more structures and proposes introduction of routines which are the same as those used in his last job.

- Analyze the situation and identify risks!

A Remark on Tools

- Typical situation: The software development is unstructured and thus not productive enough
- The (silver bullet) solution: A "new tool", e.g., a file navigator with a novel "fisheye-view".
- NB: Every tool involves structuring of product and process. The question is whether these are the right structures for the problems which must be solved and for the persons who'll use them.

Example:

- In a study about CASE-tools, several tools were compared with regard to software development productivity (function points/person-hour). Two of the tools excelled with very high productivity.
- The study also examined maintainability of the produced code. In this part of the study it appeared that one of the tools stimulated some developers to duplicate code ("cut and paste") instead of developing common (reusable) code (libraries). Consequently, maintenance became more difficult. Thus, the tool that provided structure stimulating the development of reusable code turned out to be preferable in the long run.

  NB: for the type of people that participated in the study / with their experience and training / with their tasks at hand / etc.)