Software Engineering: Theory and Practice

NMA – National Student Academy
Summer Session 2008, Nida, Lithuania

Lecture 2

Tom Verhoeff

Technische Universiteit Eindhoven
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to `T.Verhoeff@TUE.NL`

Topics

- (Software) Project Organization
- Requirements Engineering
- Configuration Management

Product and Process

What is important in a software project?

Historic learning path, becoming aware of the relevance of

- Product (ultimate deliverable)
- Product documentation, other intermediate artifacts, verification
- Process (how work is organized and done)
- Process documentation, verification of process

Project Management

Cycle of activities:

1. Plan: Make/change a plan

   Who does what when; write it down

2. Do: Execute the plan

3. Check: Monitor the plan

4. Act: Analyze and decide, go to 1

"Failing to plan is planning to fail."

General Problem Solving (left) and Waterfall Process (right)

1. Admit you have a problem	Business Case
2. Define the problem clearly	User Requirements
3. Understand and analyze the problem	Software Requirements
4. Outline a solution approach (blueprint)	Architectural Design
5. Construct an actual solution	Coding/Production
6. Teach the solution to others	Release/Transfer
7. Apply/adjust the solution	Operation and Maintenance
8. (Not applicable?)	Retirement

When doing work, always verify the work as soon as possible.

Software Development Process

There are many alternative ways of organizing activities.

There is not one right way.

Any clear process is better than no process.

Is more than project management (who does what when).

Answers questions: Why do it? Do what? Make what? Do it how?

Process quality affects product quality:

Process → Product Internals → Product Externals → User Experience

Why Requirements Engineering?

Important potential problem:

Building a good system but not the right system

Requirements Engineering: Concepts

- Requirement: capability or constraint that must be met in order to satisfy a contract or specification

- Quality characteristic/dimension/factor: . . . , e.g. Functionality, Reliability, Usability, Efficiency, Maintainability, Portability

How To Obtain Requirements: Elicitation and Analysis

- Interview, survey
- Study existing products
- Analyze requirements of system that contains the software
- Consolidate and structure raw requirements
- Follow-up, discuss draft requirements
- Prototype (could be a paper mock-up)
- Model (e.g. state machine)

General Requirements

- Name, purpose
- Overview of capabilities, constraints, rationale
- User categories, typical usage
- Operational environment, context diagram
- External interfaces
- Expected future changes

Specific Requirements

- Atomic (single sentence; avoid passive voice)
- Identifier (ensure traceability), priority
- Capability requirements: include capacity, speed, accuracy
- Constraint requirements: product and process qualities
- Redundancy: good and bad, cross reference
- Concern the problem domain, *not* the solution domain.
  N.B. Software engineers often lack problem domain knowledge
- Must be verifiable

Why Configuration Management?

Important potential problems:

- Ambiguity as to what document or file is meant
- Inconsistent combinations of files
- Lost files, lost changes
- Undocumented, unapproved changes
- Not knowing the composition of released software
- Not being able to go back to a previous version

Configuration Management (CM) Terminology

- Configuration Item (CI): (document, software, hardware) entity treated as a unit for CM; atomic or composite
- Version and relations between versions:
  - revision replaces obsolete/incorrect version
  - variant exists parallel to other version
- Baseline: formally reviewed and approved CI serving as a basis for further development

Configuration Management Goals

- Identify and define the CIs of the project: all relevant artifacts
- Control the release and change of CIs throughout the project
- Record and report the status of CIs and of change requests
- Verify the completeness and correctness of CIs

Configuration Management Tools

- Tools are not a complete solution: also need to use them well.

  Establish clear and effective procedures and rules. Train users.

- CVS, Subversion, Bazaar, Mercurial, Git

- Bugzilla

- Trac: an enhanced wiki and issue tracking system for software development projects

- Also consider tools for building and releasing (not treated here).

Configuration Management with Subversion

- One central repository with current and past versions
  - Trunk holds main line of development
  - Tags mark baselines, releases, . . .
  - Branches hold variants, experiments, . . .
- Multiple local working copies
  - checkout, update, commit

Copying in the repository is cheap, both in time and in memory usage.

Tagging and branching are done by copying. The name of the copy identifies the tag or branch.

Subversion: Create Working Copy from Repository

Subversion: Create Working Copy from Repository

Subversion: Commit Changes in Working Copy to Repository

Configuration Management: Good Practices

- Only commit code that compiles, runs, and passes all unit tests.
- When committing, always include a log message explaining *why* the change was made (**svn diff** can tell you *what* changed).

References

- Example: *Anagrams User Requirements Document*
- Checklist for User Requirements
- "Configuration Management", Ch. 17 from *Software Engineering: Planning for Change* by D. A. Lamb. Prentice-Hall, 1988.
- *Common CM Tasks in Subversion* by T. Verhoeff
- *The Subversion Book*
- Turtoise SVN: GUI client for Subversion on Windows
- svnX: GUI client for Subversion on Mac OS X