Software Engineering: Theory and Practice

(Software) Project Organization

Requirements Engineering

Tom Verhoeff

Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to `T.Verhoeff@TUE.NL`

Topics

- (Software) Project Organization
- Requirements Engineering

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

References

- Example: *Anagrams User Requirements Document*
- Checklist for User Requirements
- Article: "The Top Risks of Requirements Engineering"
- Article: "Requirements and Testing: Seven Missing-Link Myths"
- Article: "SMART Requirements"
- Book: *Software Requirements and Specifications: A Lexicon of Practice, Principles, and Prejudices* by Michael Jackson