Lecture 3 - Software Process

- Introduction

• Why do we need to be careful about software processes?
  – The “machines” are human: creative, unreliable, sickly, mercenary, etc.
  – Requirements will keep changing.
  – Complexity of the system, and the development environment.
  – Mistakes are monumentally expensive.

• Outsourcing and CMM

• Elements of a software process

• Types of Software Processes
  – Ad-Hoc: When novelty, time-to-market are key
  – Waterfall: Established, stable, safety-critical, when requirements are clear up front etc.
  – Prototype: When requirements are not crystal-clear up-front, e.g., with new applications where human interaction is key.
  – New ideas: Open Source, and XP

• Applicability of processes: Examples.
Process Elements

1. The Stake Holders:
   - *client* who puts up the bucks.
   - *user* the person who uses the software.
   - *developers* who is building it.

2. The Process Description: Process Description; could be formal, written down, repeatable, or just informal & part of the “culture”.

3. Constituent Steps: Different stages that the project will go through (next slide)

4. Intermediate Products: various documents/prototypes and 4)

5. Controls How is controlled? Measured? How are problems found and fixed..

Analogy to manufacturing..
Steps in Process

• **Goals** why this step?

• **Participants** Who are players?

• **Models/Methods/Tools** What can we use to make this go well.

• **Products** What comes out when it’s done?

• **Issues/Problems** What are the issues and problems that arise in this phase?
Requirement

- **Goals** *Answer:* What are we going to build?

- **Participants** Marketers, Users, Engineers, Analysts.

- **Models/Methods/Tools** Use Case analysis; focus groups; formal specification languages; scenarios;

- **Products** A requirements/specification document: Should be *understandable, precise, complete, consistent, modifiable, and unambiguous*

- **Issues**
  - Functional/Non Functional Requirements
  - Preparation of User Manual
  - System Test Descriptions
  - Requirements on details of the software process.
High Level Design/Architecture

• **Goals** *Answer:* How will we structure the system?

• **Participants** Architects, Marketers, Performance Experts.

• **Models/Methods/Tools** Architecture Modeling languages (CORBA IDL, Wright, etc), performance models/analysis tools, boxes & arrows pictures

• **Products** A high-level design document: describes processes, major components (DB, TM, GUI etc), dependencies, rates, etc.

• **Issues**
  
  – Architectural Styles.
  – High-level trade-offs: Flexibility vs. Efficiency, etc.
  – Personnel/Organizational issues in design.
Low Level Design/Architecture

• **Goals** *Answer:* What modules/objects/frameworks will we build?

• **Participants** OO Designers, Architects, hotshot programmers.

• **Models/Methods/Tools** UML, OMT, Rational Rose, etc.

• **Products** Low level design documents, UML, C++/C header files.

• **Issues**
  – Styles: design patterns, algorithms, data structures.
  – Trade-offs Space vs. Time, etc.
  – Reuse of frameworks, libraries, etc.
Implementation

• **Goals** Crank the code, have no life.

• **Participants** Cold-shot programmers (innocent victims with B.S. Degrees, and no experience)

• **Models/Methods/Tools** Code Models/Personal software process/ Microsoft Press methods/Visual C++, Java Workshop, g++ etc.

• **Products** Code.

• **Issues**
  – Coding Standards and Practices
  – Defensive Coding: assertions, checks etc.
Testing/Verification

• **Goals** Find bugs in the system!

• **Participants** Testers, statisticians, etc.

• **Products** Tested System

• **Models/Methods/Tools** Abstract Flow Models of Code; Coverage Testing, PureCov, and other tools.

• **Issues**
  
  – Cost of testing—test oracles
  – Testing vs. Inspection vs. Formal Verification.
  – Black Box vs. Whitebox testing.
Waterfall Model
Good (⊕) & Bad (⊖)

• ⊕ Managers can keep on top of things.

• ⊕ Lots of paper: Good organizational memory.

• ⊕ Managing/avoiding conflict among stakeholders.

• ⊕ Less risk of the “ooops” factor.

• ⊖ Rigid, time consuming.

• ⊖ May miss time to market.

• ⊖ Could be down a Lot of $$$ before revenue starts.

• ⊖ Information may not be available when needed.

• ⊖ Things may change after documented.
Prototype/Evolutionary Model
Good (⊕) & Bad (⊖)

- ⊕ Customers get quick look at product.
- ⊕ Possible revenue early.
- ⊕ Able to start tuning product to market.
- ⊕ More fun for engineers.
- ⊕ Competitors may find out!
- ⊕ Competitors may find out!
- ⊖ Can be chaotic
- ⊖ How do you build a prototype quickly?
- ⊖ Infra structure available/expensive? simulation necessary?
- ⊖ “Throw away mentality” might hurt quality.
Ad-Hoc Model

Two steps:

1. Do stuff (write code mostly)
2. Ship!

- ⊕ Less bureaucratic, less document-intensive
- ⊕ No one knows product except developers.
- ⊖ No one knows product except developers.
- ⊖ Very poor quality control.
- ⊖ Difficult to diagnose problems and improve.
Why processes may fail

• Not well described, not repeatable, not well followed.

• Actual Participants $\rightarrow$ process stakeholder mapping fails.. why?

• Intermediate steps/documents not needed, not meaningful.

• The business environment moves at a different rate than the process.

• The implementation technology is not compatible with the process, or favours another process.

• Company culture incompatible with process. (HiRel, Embedded,: 1mtg/1Loc. Compare with Valley companies)
Some Examples

1. A web-based educational adventure game for kids
2. A web-based office-automation (e.g., purchasing)
3. Virus Scanning software
4. Turbo Tax.
5. Re-implement an embedded Controller for a Cruise Missile for a new processor.
6. An extremely novel, innovative "shoot-em-up" video game.
7. A Fly-by-wire system for a fighter plane.
8. Laptop Software to Control a Robot Mouse
10. Backup software for new type of Read/Write 1 Terabyte DVD drive
11. Microsoft Internet Explorer