Software Process

Overview

• What is software process?
• Examples of process models
• Unified Process (UP)
• Agile software development

Software Process

• Definition [Pressman]
  – a framework for the tasks that are required to build high-quality software.
  – to provide stability, control and organization to an otherwise chaotic activity

Code-and-Fix Process

• The first thing people tried in the 1950s
  1. Write program
  2. Improve it (debug, add functionality, improve efficiency, ...)
  3. GOTO 1
• Works for small 1-person projects and for some CS course assignments

Problems with Code-and-Fix

• Poor match with user needs
• Bad overall structure - No blueprint
• Poor reliability - no systematic testing
• Maintainability? What’s that?
• What happens when the programmer quits?
A More Advanced Process

Examples of Process Models

- Waterfall model
- Prototyping model
- Spiral model
- Incremental model

Waterfall Model

- The "classic" process model since 1970s
  - Also called "software life cycle"

- Analysis: Define problems
  - requirements, constraints, goals and domain concepts

- Design: Establish solutions
  - System architecture, components, relationship

- Implementation: Implement solutions

- Testing and integration: Check solutions
  - Unit testing, system testing

- Maintenance: the longest phase

Key Points of the Model

- The project goes through the phases sequentially

- Possible feedback and iteration across phases
  - e.g., during coding, a design problem is identified and fixed

- Typically, few or no iterations are used
  - e.g., after a certain point of time, the design is "frozen"

Waterfall Model Assumptions

- All requirements are known at the start and stable

- Risks (unknown) can be turned into known through schedule-based invention and innovation

- The design can be done abstractly and speculatively
  - i.e., it is possible to correctly guess in advance how to make it work

- Everything will fit together when we start the integration
Pros and Cons

- **Pros:**
  - Widely used, systematic, good for projects with well-defined requirements
  - Makes managers happy
- **Cons:**
  - The actual process is not so sequential
  - A lot of iterations may happen
  - The assumptions usually don't hold
  - Working programs are not available early
  - High risk issues are not tackled early enough
  - Expensive and time-consuming

When would you like to use waterfall?

- Work for big clients enforcing formal approach on vendors
- Work on fixed-scope, fixed-price contracts without many rapid changes
- Work in an experienced team

Observation

- Top three reasons for at least partial failure projects
  - Lack of user input
  - Incomplete requirements, and
  - Changing requirements

Prototyping Model

- Build a prototype when customers have ambiguous requirements

Key Points of the Model

- Iterations: customer evaluation followed by prototype refinement
- The prototype can be paper-based or computer-based
- It models the entire system with real data or just a few screens with sample data
- Note: the prototype is thrown away!

Pros and Cons

- **Pros**
  - Facilitate communication about requirements
  - Easy to change or discard
  - Educate future customers
- **Cons**
  - Iterative nature makes it difficult to plan and schedule
  - Excessive investment in the prototype
  - Bad decisions based on prototype
    - E.g., bad choice of OS or PL
When would you like to use prototyping?

• When the desired system has a lot of interactions with users

Spiral Model

• A risk-driven evolutionary model that combines development models (waterfall, prototype, etc.)

Spiral Phases

• Objective setting
  – Define specific objectives, constraints, products, plans
  – Identify risks and alternative strategies
• Risk assessment and reduction
  – Analyze risks and take steps to reduce risks
• Development and validation
  – Pick development methods based on risks
• Planning
  – Review the project and decide whether to continue with a further loop

What Is Risk?

• Something that can go wrong
  – People, tasks, work products
• Risk management
  – Risk identification
  – Risk analysis
    • the probability of the risk, the effect of the risk
  – Risk planning
    • various strategies
  – Risk monitoring

Risk Planning [Sommerville]

<table>
<thead>
<tr>
<th>Risk</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recruitment problems</td>
<td>Alert customer of potential difficulties and the possibility of delay, investigate buying-in-components</td>
</tr>
<tr>
<td>Defective components</td>
<td>Replace potentially defective components with bought-in components of known reliability</td>
</tr>
<tr>
<td>Requirements changes</td>
<td>Derive traceability information to assess requirements change impact, maximize information hiding in the design</td>
</tr>
<tr>
<td>Organizational financial problems/</td>
<td>Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business</td>
</tr>
<tr>
<td>restructuring</td>
<td>Investigate buying-in components, investigate the use of a program generator</td>
</tr>
<tr>
<td>Underestimated development time</td>
<td></td>
</tr>
</tbody>
</table>

Key Points of the Model

• Introduce risk management into process
• Develop evolutionary releases to
  – Implement more complete versions of software
  – Make adjustment for emergent risks
Pros and Cons

• Pros
  – High amount of risk analysis to avoid/reduce risks
  – Early release of software, with extra functionalities added later
  – Maintain step-wise approach with “go-backs” to earlier stages

• Cons
  – Require risk-assessment expertise for success
  – Expensive

When to use the model?

• Large and mission-critical projects
• Medium to high-risk projects
• Significant changes are expected

Incremental Model

• A sequential of waterfall models

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Design</th>
<th>Implementation</th>
<th>Testing &amp; Integration</th>
<th>Feedback, adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Release n</td>
</tr>
</tbody>
</table>

Iteration n: 3 weeks
(for example)

Iteration n+1: 3 weeks
(for example)

Key Points of the Model

• Iterative: many releases/increments
  – First increment: core functionality
  – Successive increments: add/fix functionality
  – Final increment: the complete product

• Require a complete definition of the whole system to break it down and build incrementally

Pros and Cons

• Pros
  – Early discovery of software defects
  – Early delivery of working software
  – Less cost to change/identify requirements

• Cons
  – Constant changes (“feature creep”) may erode system architecture

When to use the model?

• The requirements of the complete system are clear
• Major requirements must be defined while some details can evolve over time
• Need to get a product to the market early
Spiral model vs. incremental model

- Iterative models
  - Most projects build software iteratively
- Risk-driven vs. client-driven

Unified Process (UP)

- An example of iterative process for building object-oriented systems
  - Very popular in the last few years
  - By the same folks who develop UML
- It provides a context for our discussion of analysis and design

Phases in UP

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Inception: preliminary investigation
- Elaboration: analysis, design, and some coding
- Construction: more coding and testing
- Transition: beta tests and development
- Each phase may be enacted in an iterative way, and the whole set of phases may be enacted incrementally

Iteration Length

- Iteration should be short (2-6 weeks)
  - Small steps, rapid feedback and adaptation
  - Massive teams with lots of communication but no more than 6 months
- Iterations should be timeboxed (fixed length)
  - Integrate, test and deliver the system by a scheduled date
  - If not possible: move tasks to the next iteration

Reasons for Timeboxing

- Improve programmer productivity with deadlines
- Encourage prioritization and decisiveness
- Team satisfaction and confidence
  - Quick and repeating sense of completion, competency, and closure
  - Increase confidence for customers and managers

UP Disciplines

- Discipline: an activity and related artifact(s)
- Artifact: any kind of work product
  - Requirement modeling
    - requirement analysis + use-case models, domain models, and specs.
  - Design
    - design + design models
  - Implementation
    - code
Agile Software Development

- A timeboxed iterative and evolutionary development process
- It promotes
  - adaptive planning
  - evolutionary development,
  - incremental delivery
  - rapid and flexible response to change

Any iterative method, including the UP, can be applied in an agile spirit.

Agile Methods

- Agile Unified Process (Agile UP)
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Scrum

Agile UP

- Keep it simple
  - Prefer a small set of UP activities and artifacts
  - Avoid creating artifacts unless necessary
- Planning
  - For the entire project, there is only a high-level plan (Phase Plan), to estimate the project end date and other major milestones
  - For each iteration, there is a detailed plan (Iteration plan) created one iteration in advance

The Agile Manifesto

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions over Processes and tools
  - Working software over Comprehensive documentation
  - Customer collaboration over Contract negotiation
  - Responding to change over Following a plan

Key Points of Agile Modeling

- The purpose of modeling is primarily to understand, not to document
- Modeling should focus on the smaller percentage of unusual, difficult, and tricky parts of the design space
- Model in pairs (or triads)
- Developers should do the OO design modeling for themselves
- Create models in parallel
  - E.g., interaction diagram & static-view class diagram

Models are inaccurate

- Only tested code demonstrates the true design
- Treat diagrams as throw-away explorations
- Use the simplest tool possible to facilitate creative thinking
  - E.g., sketching UML on whiteboards
- Use "good enough" simple notation
Pros and Cons

• Pros
  – Customer satisfaction by rapid, continuous delivery of useful software
  – Close, daily cooperation between business people and developers
  – Better software quality and lower cost

• Cons
  – People may lose sight of the big picture
  – Heavy client participation is required
  – Poor documentation support for training of new clients/programmers

When to use agile methods?

• Changing requirements
• Faster time to market and increased productivity
• Frequently used in start-up companies