Software Processes

Chapter 2

Software Processes

Outline:

1. What is a software process?
   A. Four primary software engineering activities
2. Traditional software process models
   A. Waterfall
   B. Incremental development
   C. Spiral model
   D. Reuse-oriented software engineering
3. Coping with change
4. Example process: Rational Unified Process

1. What is a software process?

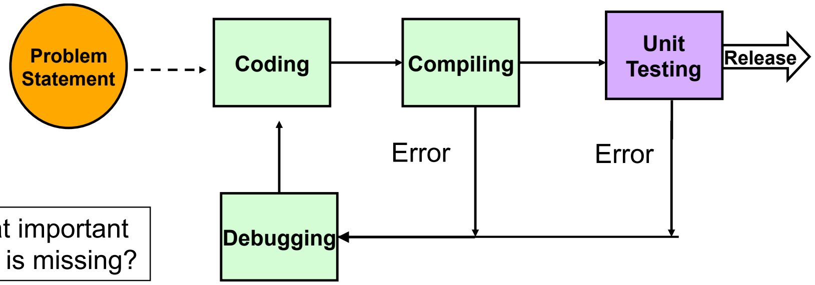
- Software Process: A structured set of activities used to develop a software system.

- It is a description of
  - what tasks need to be performed in
  - what sequence under
  - what conditions by
  - whom to

  achieve the “desired results.”

- Desired results: high quality software product.

A simple process



What important step is missing?

- Suitable for student projects
- Students encounter problems when
  - some steps are skipped
  - problem statement is not well stated or understood

As projects get larger and more complex . . .

- We need more people and more coordination
  - Problem statement needs to be expanded and clarified (requirements/specifications)
  - Need a good, well-documented design
  - Need to make sure various developers can work together (tools, documentation)
  - Need to ensure adequate testing is done
- We need a more detailed process

1A. Four primary software engineering activities

- There are many different software processes but all involve these activities:
  - **Specification** – defining what the system should do (stating the requirements)
  - **Development** – defining the organization of the system (aka the design) and implementing the system
  - **Validation** – checking that the system does what the customer wants
  - **Evolution** – changing the system in response to customer needs.
- Different software processes do the activities in different ways.

Software specification

- The process of establishing the requirements:
  - the features/functions that are required by the users
  - the constraints on the system's operation and development.
- Requirements engineering process
  - Requirements elicitation and analysis
    - ❖ What do the customers/users require or expect from the system?
    - ❖ May observe existing systems, develop models or prototype
  - Requirements specification
    - ❖ Defining the requirements in detail and documenting them.
  - Requirements validation
    - ❖ Checking them for clarity, consistency, completeness, etc.

Software development:
design and implementation

- Converting the requirements into an executable system.
- Software design
  - Description of the structure of the software using various models (describing the subcomponents, how they interact, etc)
- Implementation
  - Translate the design into an executable program
- Design and implementation are closely related and often interleaved.

Software validation

- Verification and validation (V & V) is intended to
  - show that a system conforms to its specification and
  - meets the needs of the system customer.
- Program testing:
  - executing the system over simulated data, ensuring the results are correct.
- Inspections and reviews:
  - humans analyze models and source code looking for errors or problems

Software evolution

- After the software has been released, it must be kept up to date.
  - Customers require new functions
  - Defects must be repaired
  - Must adapt to new platforms and machines
- Activities include:
  - Modifying requirements/specifications (as needed)
  - Modifying design (as needed)
  - Modifying the implementation
  - Retesting, adding new test cases.

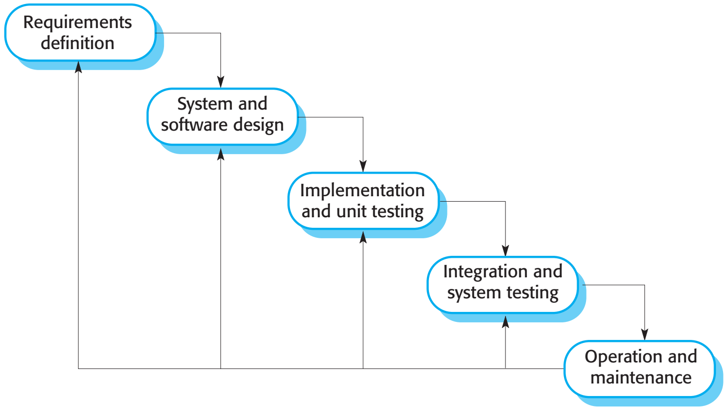
2. Traditional Software process models
(or frameworks, or paradigms)

- A software process model:
  - is a simplified (or abstract) representation of a set of specific software processes.
  - must be “extended” with more detail to become an actual software process.
- Traditional software process models:
  A. Waterfall model
  B. Incremental development
  C. Spiral model
  D. Reuse-oriented software engineering

2A. Waterfall model

- The waterfall model
  - One of the first published models
  - Separate and distinct phases are performed in sequence.
  - Planning occurs up front: “Plan-driven”
- The separate phases:
  - Requirements definition
  - Software design
  - Implementation
  - Testing
  - Maintenance
- The output of one stage is input to the next.
- Tends to require/generate much documentation.

Waterfall model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

What makes it go backwards?

Waterfall model issues

- Good features:
  - Simple and easy to implement (better than no process)
  - Easy for managers to track the progress of the project
- Can be used for large projects when a system is developed at several sites.
  - Plan-driven nature of the this model helps coordinate the work.
- Main drawback: The difficulty of accommodating changes after the process is underway.
  - Change requires "backtracking": revising previous step(s), re-work (costly)
  - This model is appropriate only when
    - a) the requirements are well-understood upfront and
    - b) changes will be fairly limited during the design process.
- Customers often need to change the requirements

2B. Incremental development

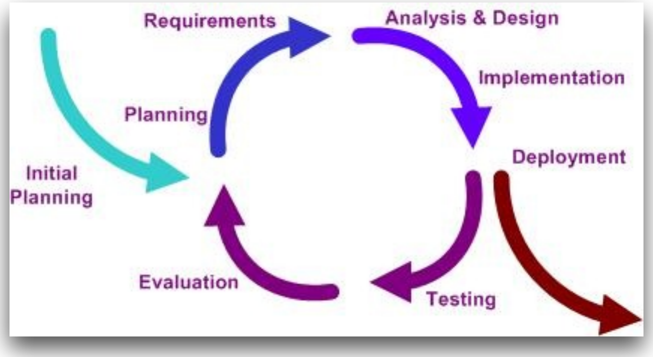
(a.k.a Iterative development)

- Several development iterations are performed in sequence.
- Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test

From: Craig Larman,
Agile and Iterative Development - A Manager's Guide

- Each iteration produces a new version (called an increment).
  - Each version adds functionality to the previous version.
  - Only the final version is a complete system.
- Each version is exposed to the user for feedback
  - The customer may come to the developers' site for demos/testing.
  - If the intermediate versions are given to the customer, it is called **Incremental Delivery**.

Incremental development



Each time around the loop produces a new version of the software.

Incremental development benefits

- Reduces cost of accommodating changing customer requirements.
  - Early versions are incomplete, so less re-work to do.
  - May require no changes to current version (add to future version).
- It is easier to get customer feedback.
  - Users understand a working incremental release better than documents from the specification or design phase.
- Does not need to be planned entirely up front.
- Early versions can implement the most important, urgent, or risky features

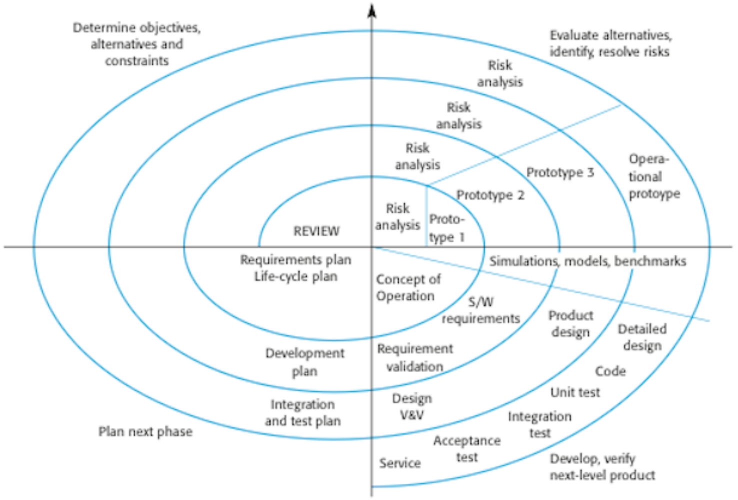
Incremental development problems

- The process is not visible
  - there's less process documentation, so it's difficult to measure progress.
  - may not know how many more increments are required.
- Difficult to design and implement common facilities needed by all versions
- System structure tends to degrade as new increments are added.
  - this makes the code more difficult to modify each time.
  - UNLESS time and money are spent on **refactoring** to improve the software.
  - **Refactoring**: disciplined technique for restructuring an existing body of code, altering its internal structure <u>without changing its external behavior</u>.
  - Modifying a program to improve its structure, reduce its complexity, or make it easier to understand.

2C. Spiral model

- Proposed by Barry Boehm in 1988.
- Process represented as a spiral
  - Each loop represents a phase in the process.
  - Content of each phase is not predetermined, plan as you go.
- Risks are explicitly assessed and resolved.
  - Assumes need for change are a result of project risks.
- Sectors of the model:
  - Identify objectives, alternatives and constraints.
  - Evaluate and reduce risk (may develop prototype).
  - Development and validation
  - Plan next phase (after review of current phase).

Figure 2.11 Boehm's spiral model of the software process



Spiral model issues

- Good for high-risk projects.
  - Often used in combination with other process models.
- In practice, the model is rarely used as published.
- Somewhat similar to incremental development, but
  - Risk assessment is incorporated into the process
  - Development is not required to be incremental:
    - prototypes and results of previous loops can be discarded.
    - production development could be postponed until the last loop.

2D. Reuse-oriented software engineering

- The system is assembled from existing components.
- Components may be in the form of
  - source code that must be compiled into the final product OR
  - already compiled code that can be accessed from other programs.
- Process stages:
  - Requirements specification (similar to other process models)
  - Component analysis: search for close matches to requirements
  - Requirements modification: to reflect available components
  - System design with reuse: organize framework around acceptable components (may require designing new code).
  - Development and integration: components are integrated along with new code
  - System validation (similar to other process models)

Types of software components for reuse

- Web services (or “API”)
  - Various “functions” available for remote invocation from apps
  - Examples: Weather API from Weather Channel, Endicia Label Server API (labels with USPS postage)
- Library of Classes: framework
  - Developed as a package to be integrated (compiled) with a component framework such as .NET or J2EE.
  - Example: parsekit for Mac OS X apps (scanners/parsers)
- Stand-alone software systems that are configured for use in a particular environment.
  - often called COTS: “Commercial off the Shelf” systems
  - Example: PeopleSoft, HR management for companies

Advantages and Disadvantages of Reuse-oriented Software Engineering

- Benefits
  - Reduces costs and risks (less code to write, already tested)
  - Usually leads to faster delivery.
- Disadvantages
  - Requirements may have to be compromised (no good matches found)
  - Control over evolution of system is lost (dependent on developers of the components).

3. Coping with change

- Change is inevitable in all large software projects.
  - Changing business environments lead to changing requirements
    - New opportunities and technologies
    - Changing markets, new competitors
  - New technologies open up new possibilities for improving implementations
  - New platforms require application changes
- Change leads to rework:
  - new requirements lead to more requirements analysis
  - this may lead to redesign of the system or components
  - this may lead to changes to the implementation
  - this may lead to new tests, and re-testing the system

Reducing the costs of rework: two approaches

- **Change avoidance:** include process activities that anticipate possible changes before significant rework is required.
  - i.e. develop a prototype to show some key features of the system to users, let them refine requirements before committing to them.
- **Change tolerance:** design the process to accommodate change at low cost
  - Use incremental development, get feedback from users.
  - Changes likely apply to most recent increment only, OR
  - Can be incorporated into later increments.

Software prototyping

- Prototype: an initial, incomplete, version of a system used to demonstrate concepts and try out options.
- Allows users to see how well system could support their work
- May lead to new ideas for requirements
- As prototype is developed, may reveal errors and omissions in the requirements
- Can check feasibility of design
  - For a database, make sure it efficient for most common queries
  - For a user interface, user understands a prototype much better than a text description (get better feedback).

Prototype process

- Objectives for prototype should be made in advance
  - Decide what to put in, what to leave out.
- Must be developed quickly!
- Users test the prototype and evaluate it with respect to the objectives
- Prototypes should be discarded after use!
  - It may be impossible to tune the prototype to meet performance and reliability requirements
  - Prototypes are normally undocumented
  - The structure is usually degraded through quick and dirty design
  - The prototype probably will not meet normal organizational quality standards.

Incremental delivery

- Special case of Incremental Development where each version is delivered to users.
- Generally same advantages as Incremental Development
  - Good response to changing requirements
- Major system functionality is available to users earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Highest priority requirements are included in early increments, so they receive the most testing.

Incremental delivery problems

- Generally same problems as Incremental Development
  - Difficult to design and implement common facilities needed by all versions
  - Constant upgrading can degrade structure of code
- Contract negotiations are more difficult
  - The specification is developed in stages
  - Unable to use it as part of the development contract.
- Difficult to get feedback when replacing an existing system:
  - Users won’t be motivated to use the less functional new system.

4. The (Rational) Unified Process

- Unified Process: A popular software process
  - a hybrid process: iterative/incremental AND staged.
- Has 4 main phases or stages.
  - correspond to business concerns, not technical activities
- Each phase may contain several iterations.
- Has 6 disciplines (= activities) performed across the 4 phases.
- Each phase involves all the disciplines, in varying amounts.

Four phases of UP

- INCEPTION
  - High level requirements established
  - Key risks identified
- ELABORATION
  - Significant elements (core architecture) are programmed and tested
- CONSTRUCTION
  - Remainder of system is built and tested
- TRANSITION
  - The system is fully deployed to the customer

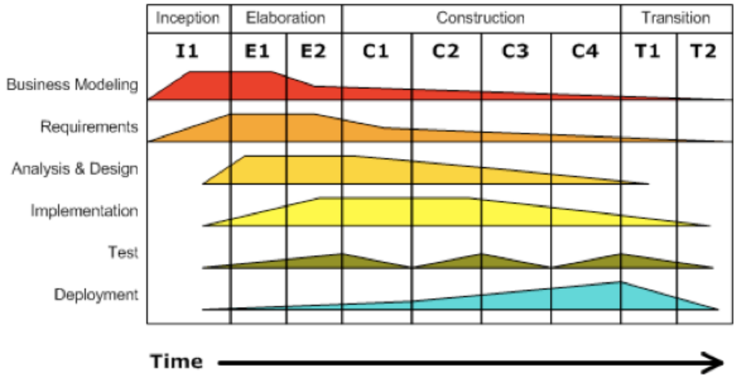
Certain milestones must be completed in each phase, before moving on to the next one.

Disciplines of UP

- Business Modeling
  - business processes are modeled using use cases
- Requirements
- Design
- Implementation
- Testing
- Deployment
  - product is released, distributed, and installed
- Project Management
  - scheduling, managing resources

Phases of UP

- Disciplines over the phases
  - each column is an iteration.



The Rational Unified Process

- Rational Unified Process (RUP) is a refinement or specialization of UP
  - A product from IBM
  - Hyperlinked knowledge base with sample artifacts
  - Enables developer organization to tailor UP to its needs:
    - allows developer to select appropriate elements of the process
    - manages documentation
    - provides tools for applying the process