Software Processes

Chapter 2

Software Processes
in the textbook

- 2.1 Software process models
- 2.2 Process activities
- 2.3 Coping with change
  - Skipping 2.3.3 Boehm's spiral model
- 2.4 The Rational Unified Process
  - An example of a modern software process.

A software process

- A structured set of activities used to develop a software system/product.
- Many different software processes but all involve these activities:
  - **Specification** – defining what the system should do (requirements)
  - **Development** – defining the organization of the system (design) and implementing the system
  - **Validation** – checking that it does what the customer wants;
  - **Evolution** – changing the system in response to changing customer needs.
- A software process model (or paradigm) is an abstract representation of a software process
  - specific processes are derived from each model by adding details

2.1 Software process models (or frameworks, or paradigms)

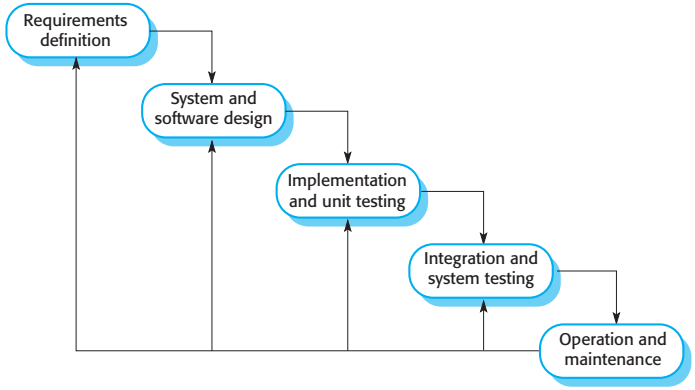
- The waterfall model
  - Separate and distinct phases of specification, development, validation and evolution, performed in sequence.
  - Planning occurs upfront: "Plan-driven"
- Incremental development
  - Specification, development and validation are interleaved in cycles, producing a series of versions of the software.
- Reuse-oriented software engineering
  - The system is assembled from existing components.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

Waterfall model phases

- There are separate identified phases:
  - Requirements definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Main drawback: The difficulty of accommodating change after the process is underway.
  - In principle, a phase has to be complete before moving on to the next phase.
  - Change requires “backtracking”: revising previous step(s), re-work

Waterfall model



What makes it go backwards?

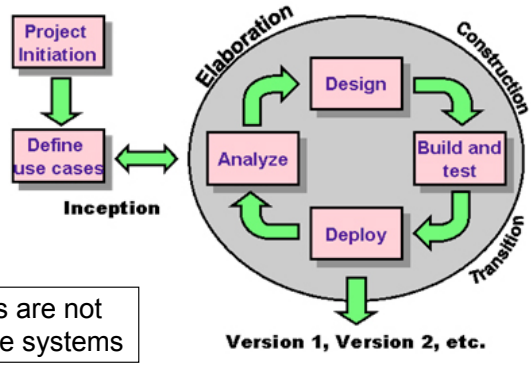
Waterfall model issues

- Partitioning the project into sequential stages makes it difficult to respond to changing customer requirements.
  - Appropriate only when
    a) the requirements are well-understood and
    b) changes will be fairly limited during the design process.
- Can be used for large systems engineering projects where a system is developed at several sites.
  - Plan-driven nature of the this model helps coordinate the work.
- Good when formal methods of software development are required.
  - Formal methods: using a mathematical model of system specifications and refining it to programming language code using transformations
  - Good when safety, reliability, and security requirements are critical.

Incremental software development

- Specification, development and validation are interleaved in cycles.
- The system is developed as a series of versions or releases (called increments).
  - 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
  - If the intermediate versions are given to the customer(s), it is called **Incremental Delivery**.
- Early versions can implement the most important, urgent, or risky features

Incremental development



Versions are not complete systems

Feedback from use of each version is incorporated into next Analyze phase

Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - 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 on the work that has already been done.
  - Easier to present a working incremental release than results of specification or design phase.
- Can be plan-driven (all versions planned ahead) OR plan each increment as it is encountered.

Incremental development problems

- The process is not visible
  - generally less process documentation, so it's difficult to measure progress.
- System structure tends to degrade as new increments are added.
  - 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 without changing its external behavior.
  - Modifying a program to improve its structure, reduce its complexity, or make it easier to understand.

Reuse-oriented software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Requirements specification
  - Component analysis: search for close matches
  - Requirements modification: to reflect available components
  - System design with reuse: organize framework around acceptable components.
  - Development and integration: components are integrated along with new code
  - System validation

Types of software components

- Web services
  - Developed according to service standards
  - Are available for remote invocation from web apps or clients
  - Example: Google maps, Amazon web services
- 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 (COTS) that are configured for use in a particular environment.
  - Example: PeopleSoft, HR management for companies

Advantages and Disadvantages of Reuse-oriented Software Engineering

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

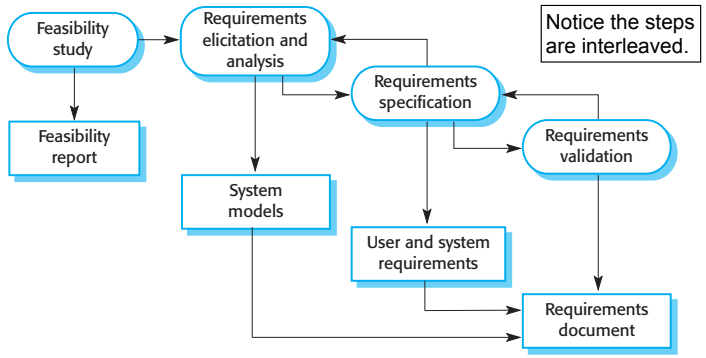
2.2 Process activities

- The four basic process activities:
  - specification
  - development
  - validation
  - evolution
- organized differently in different development processes. (i.e. in sequence or inter-leaved).
- Same activity may be carried out differently by different people, or different process methods (i.e. specifications can be typed into a document or written on cards).

Software specification

- The process of establishing the requirements:
  - the services that are required by the users (features/functions) and
  - the desired constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
    - May observe existing systems, develop models or prototype
  - Requirements specification
    - Defining the requirements in detail, write up in a document
  - Requirements validation
    - Checking the requirements for realism, consistency, and completeness.

The requirements engineering process



Software Development:
design and implementation

- Converting the system specification into an executable system.
- Software design
  - Description of the structure of the software, data models, interfaces, algorithms, etc.
- Implementation
  - Translate the design into an executable program
- Design and implementation are closely related and may be inter-leaved.

Design activities

- **Architectural design**: where you identify
  - the overall structure of the system,
  - the principal components,
  - their relationships and
  - how they are distributed.
- **Interface design**, where you precisely define the interfaces between system components (how the communicate) (so they can be developed independently).
- **Component design**, where you design how each component will function
- **Database design**, where you design the system data structures and how these are to be represented in a database.

Software validation

- Verification and validation (V & V) is intended to
  - show that a system conforms to its specification and
  - meets the requirements of the system customer.
- Program testing is the principal technique.
  - executing the system over simulated data
- Validation may also involve inspections and reviews
  - humans analyze models and source code looking for errors or problems

Testing stages

- Development testing
  - Parts of the system are tested independently by developers
  - Unit testing: individual program units or classes are tested
  - Component testing: coherent groupings of functions or objects are tested
  - System testing: testing of the system as a whole after integrating the components.
- Release testing:
  - a separate testing team tests a complete version of the system before it is released to users.
- User testing:
  - users or potential users of a system test the system in their own environment.

Software evolution

- Software must change to remain useful
  - The business environment changes (new functions required)
  - Errors must be repaired
  - New computers and equipment are added to the system
  - The performance or reliability may have to be improved.
- Key problem: managing change to existing software systems
- Activities include:
  - Modifying requirements/specifications
  - Modifying design
  - Modifying the implementation
  - Retesting, adding new test cases.

2.3 Coping with change

- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing 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

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

Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- Allows users to see how well system supports 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.

Prototype development process

- Objectives for prototype should be made in advance
- Decide what to put in, what to leave out.
  - But must be developed quickly
- Let users test the prototype and evaluate it with respect to the objectives

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet performance and reliability requirements
  - Prototypes are normally undocumented
  - The prototype structure is usually degraded through quick and dirty design
  - The prototype probably will not meet normal organizational quality standards.

Incremental delivery

- Incremental development where each version is delivered to users, deployed in their environment(s).
- Highest priority requirements are included in early increments.
- Requirements are frozen for the current increment, though requirements for later increments can continue to evolve.

Incremental delivery advantages

- 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.
- The highest priority system services tend to receive the most testing, since they are developed first.

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 the complete system specification as part of the development contract.
- Difficult to replace an existing system:
  - Early versions have much less functionality than the system being replaced, so users won't be motivated to use the less functional new system.

2.4 The (Rational) Unified Process

- Unified Process: A popular iterative process framework
  - A good example of a hybrid software process model
- Rational Unified Process (RUP) is a refinement or specialization of UP
  - A product from IBM
  - Enables developer organization to tailor UP to its needs, manages documentation, etc.
- UP has 6 disciplines (activities) performed over 4 phases.
- Each phase may have several iterations

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

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

Phases of UP

- Disciplines over the phases