Software Processes
Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution
- To explain the Rational Unified Process model
- To introduce CASE technology to support software process activities
The software process

● A structured set of activities required to develop a software system
  ✓ Specification – defining what the system should do;
  ✓ Design and implementation – defining the organization of the system 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 is an abstract representation of a process. It presents a description of a process from some particular perspective.
Software process descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.

- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.
Plan-driven and agile processes

- **Plan-driven** processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

- In **agile** processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.

- In practice, most practical processes include elements of both plan-driven and agile approaches.

- There are no right or wrong software processes.
Software process models

- **The waterfall model**
  - Plan-driven model. Separate and distinct phases of specification and development.

- **Incremental development**
  - Specification, development and validation are interleaved. May be plan-driven or agile.

- **Reuse-oriented software engineering**
  - The system is assembled from existing components. May be plan-driven or agile.

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

1. Requirements definition
2. Requirements analysis and definition
3. System and software design
4. Implementation and unit testing
5. Integration and system testing
6. Operation and maintenance
Waterfall model

- **Analysis**
  - Tasks: specify requirements, determine various components of task

- **Design**
  - Tasks: specify algorithmic solution, describe data structure and methods

- **Implementation**
  - Tasks: implement the data structure and the algorithms

- **Testing**
  - Tasks: test various components at various levels for correctness

- **Maintenance**
  - Tasks: debugging, adapting, enhancing the software
Waterfall model problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.
Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
Evolutionary/Incremental development

- **Exploratory development**
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.

- **Throw-away prototyping**
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.
Evolutionary/Incremental development

Outline description

Concurrent activities

Specification

Development

Validation

Initial version

Intermediate versions

Final version
Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.

- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.
Incremental development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.
Evolutionary/Incremental development

- **Problems**
  - Lack of process visibility;
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.

- **Applicability**
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.
Component-based software engineering/
Reuse-oriented software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.

- Reuse is now the standard approach for building many types of business system
  - Reuse covered in more depth in Chapter 16.
Reuse-oriented development

- Requirements specification
- Component analysis
- Requirements modification
- System design with reuse
  - Development and integration
  - System validation
Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.
Software specification

- The process of establishing what services are required and the 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?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements
The requirements engineering process

- Feasibility study
- Requirements elicitation and analysis
- Requirements specification
- Requirements validation
- System models
- User and system requirements
- Requirements document
Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.
Design process activities

- **Architectural design**, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.

- **Interface design**, where you define the interfaces between system components.

- **Component design**, where you take each system component and design how it will operate.

- **Database design**, where you design the system data structures and how these are to be represented in a database.
The software design process

Requirements specification

Design activities

Architectural design
Abstract specification
Interface design
Component design
Data structure design
Algorithm design

System architecture
Software specification
Interface specification
Component specification
Data structure specification
Algorithm specification

Design products
Programming and debugging

- Translating a design into a program and removing errors from that program.

- Programming is a personal activity - there is no generic programming process.

- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.
The debugging process

1. Locate error
2. Design error repair
3. Repair error
4. Re-test program
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.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.
The testing process

- Component testing
- System testing
- Acceptance testing
Testing stages

- **Component or unit testing**
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.

- **System testing**
  - Testing of the system as a whole. Testing of emergent properties is particularly important.

- **Acceptance testing**
  - Testing with customer data to check that the system meets the customer’s needs.
Testing phases

1. Requirements specification
2. System specification
3. System design
4. Detailed design
5. Module and unit code and test
6. Acceptance test plan
7. System integration test plan
8. Sub-system integration test plan
9. Acceptance test
10. System integration test
11. Sub-system integration test

Software engineering Lecture Notes 02 30
Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.
System evolution

1. Define system requirements
2. Assess existing systems
3. Propose system changes
4. Modify systems

Existing systems → New system
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 so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality
Reducing the costs of rework

• Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.

• Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.

- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore options and develop a UI design;
  - In the testing process to run back-to-back tests.
Benefits of prototyping

- Improved system usability.
- A closer match to users’ real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.
The process of prototype development

- Establish prototype objectives
- Define prototype functionality
- Develop prototype
- Evaluate prototype

Prototyping plan
Outline definition
Executable prototype
Evaluation report
Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security
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 non-functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organisational quality standards.
Process iteration

- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
- Iteration can be applied to any of the generic process models.
- Two (related) approaches
  - Incremental delivery;
  - Spiral development.
Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
Incremental development and delivery

- **Incremental development**
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.

- **Incremental delivery**
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.
Incremental delivery
Incremental delivery advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.
Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.
Boehm’s spiral model

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.
Spiral model of the software process

- Determine objectives, alternatives and constraints
- Risk analysis
- Evaluate alternatives, identify, resolve risks
- Prototype 3
- Operational prototype
- Simulations, models, benchmarks
- Product design
- Detailed design
- Code
- Unit test
- Integration test
- Develop, verify next-level product
- Acceptance test
- Integration and test plan
- Requirement validation
- Requirement plan
- Life-cycle plan
- Concept of Operation
- S/W requirements
- Prototype 1
- Prototype 2
- Risk analysis
- Review
- Plan next phase
- Design V&V
- Service
Spiral model sectors

- **Objective setting**
  - Specific objectives for the phase are identified.

- **Risk assessment and reduction**
  - Risks are assessed and activities put in place to reduce the key risks.

- **Development and validation**
  - A development model for the system is chosen which can be any of the generic models.

- **Planning**
  - The project is reviewed and the next phase of the spiral is planned.
Spiral model usage

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.
The Rational Unified Process

- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practical perspective that suggests good practice.
Phases in the Rational Unified Process

- Inception
- Elaboration
- Construction
- Transition

Phase iteration
RUP phases

- **Inception**
  - Establish the business case for the system.

- **Elaboration**
  - Develop an understanding of the problem domain and the system architecture.

- **Construction**
  - System design, programming and testing.

- **Transition**
  - Deploy the system in its operating environment.
RUP good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software
# Static workflows

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business modelling</td>
<td>The business processes are modelled using business use cases.</td>
</tr>
<tr>
<td>Requirements</td>
<td>Actors who interact with the system are identified and use cases are developed to model the system requirements.</td>
</tr>
<tr>
<td>Analysis and design</td>
<td>A design model is created and documented using architectural models, component models, object models and sequence models.</td>
</tr>
<tr>
<td>Implementation</td>
<td>The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.</td>
</tr>
<tr>
<td>Test</td>
<td>Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.</td>
</tr>
<tr>
<td>Deployment</td>
<td>A product release is created, distributed to users and installed in their workplace.</td>
</tr>
<tr>
<td>Configuration and change management</td>
<td>This supporting workflow managed changes to the system (see Chapter 29).</td>
</tr>
<tr>
<td>Project management</td>
<td>This supporting workflow manages the system development (see Chapter 5).</td>
</tr>
<tr>
<td>Environment</td>
<td>This workflow is concerned with making appropriate software tools available to the software development team.</td>
</tr>
</tbody>
</table>
RUP good practice

- Develop software iteratively
  - Plan increments based on customer priorities and deliver highest priority increments first.

- Manage requirements
  - Explicitly document customer requirements and keep track of changes to these requirements.

- Use component-based architectures
  - Organize the system architecture as a set of reusable components.
RUP good practice

- **Visually model software**
  - Use graphical UML models to present static and dynamic views of the software.

- **Verify software quality**
  - Ensure that the software meet’s organizational quality standards.

- **Control changes to software**
  - Manage software changes using a change management system and configuration management tools.
Key points

- Software processes are the activities involved in producing and evolving a software system.
- Software process models are abstract representations of these processes.
- General activities are specification, design and implementation, validation and evolution.
- Generic process models describe the organisation of software processes. Examples include the waterfall model, evolutionary development and component-based software engineering.
- Iterative process models describe the software process as a cycle of activities.
Key points

- Requirements engineering is the process of developing a software specification.
- Design and implementation processes transform the specification to an executable program.
- Validation involves checking that the system meets to its specification and user needs.
- Evolution is concerned with modifying the system after it is in use.
- The Rational Unified Process is a generic process model that separates activities from phases.
- CASE technology supports software process activities.
Independent of the models, the software development process can still be roughly divided into the following stages:

- Analysis (Requirement specification)
- Design
- Implementation
- Testing
- Maintenance

Can also be applied to problem solving in general!