Software Processes

- Coherent sets of activities for specifying, designing, implementing and testing software systems
Objectives

- To introduce software process models
- To describe a number of different process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution
- To introduce CASE technology to support software process activities
Topics covered

- Software process models
- Process iteration
- Software specification
- Software design and implementation
- Software validation
- Software evolution
- Automated process support
The software process

- A structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution

- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective
Software process models
Generic software process models

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

- **Evolutionary development**
  - Specification and development are interleaved

- **Formal systems development**
  - A mathematical system model is formally transformed to an implementation

- **Reuse-based development**
  - The system is assembled from existing components
Waterfall model

Requirements definition → System and software design

System and software design → Implementation and unit testing

Implementation and unit testing → Integration and system testing

Integration and system testing → Operation and maintenance

©Ian Sommerville 2000
Software Engineering, 6th edition. Chapter 1
Slide 7
Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway
Waterfall model problems

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood
Evolutionary 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

- **Throw-away prototyping**
  - Objective is to understand the system requirements. Should start with poorly understood requirements
Evolutionary development

Concurrent activities

Outline description

Specification

Development

Validation

Initial version

Intermediate versions

Final version
Evolutionary 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
Formal systems development

- Based on the transformation of a mathematical specification through different representations to an executable program
- Transformations are ‘correctness-preserving’ so it is straightforward to show that the program conforms to its specification
- Embodied in the ‘Cleanroom’ approach to software development
Formal systems development

- Requirements definition
- Formal specification
- Formal transformation
- Integration and system testing
Formal transformations

Formal specification

R1

P1

R2

P2

R3

P3

Executable program

P4

Proofs of transformation correctness
Formal systems development

- **Problems**
  - Need for specialised skills and training to apply the technique
  - Difficult to formally specify some aspects of the system such as the user interface

- **Applicability**
  - Critical systems especially those where a safety or security case must be made before the system is put into operation
Reuse-oriented development

- 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

- This approach is becoming more important but still limited experience with it
Reuse-oriented development
Process iteration
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 development
  - Spiral development
Incremental development

- 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

Define outline requirements → Assign requirements to increments → Design system architecture

Develop system increment → Validate increment → Integrate increment → Validate system

Final system

System incomplete
Incremental development 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
Extreme programming

- New approach to development based on the development and delivery of very small increments of functionality
- Relies on constant code improvement, user involvement in the development team and pairwise programming
Spiral development

- 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
- Evaluate alternatives, identify, resolve risks
- Plan next phase
- Review
- Plan next phase
- Prototype 1
- Prototype 2
- Prototype 3
- Simulation, models, benchmarks
- Operational prototype
- Detailed design
- Code
- Integration test
- Acceptance test
- Integration and test plan
- Development plan
- Concept of operation
- Simulations, models, benchmarks
- Requirement validation
- S/W requirements
- Design V&V
- Acceptance test
- Service
- Develop, verify next-level product
- Next-level product
- Service
- Develop, verify next-level product
- Plan next phase
- Review
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
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
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation
The requirements engineering process

1. Feasibility study
2. Requirements elicitation and analysis
3. System models
4. User and system requirements
5. Requirements specification
6. Requirements validation
7. Requirements document

Feasibility study leads to Requirements elicitation and analysis, which leads to System models. System models lead to User and system requirements, which lead to Requirements specification, which leads to Requirements validation, which leads to 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
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design
The software design process

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

Design activities

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

Design products
Design methods

- Systematic approaches to developing a software design
- The design is usually documented as a set of graphical models
- Possible models
  - Data-flow model
  - Entity-relation-attribute model
  - Structural model
  - Object models
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

- Locate error
- Design error repair
- Repair error
- Re-test program
Software validation

- Verification and validation 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.
The testing process

Component testing  Integration testing  User testing

Unit testing  Module testing  Sub-system testing  System testing  Acceptance testing
Testing stages

- **Unit testing**
  - Individual components are tested

- **Module testing**
  - Related collections of dependent components are tested

- **Sub-system testing**
  - Modules are integrated into sub-systems and tested. The focus here should be on interface testing

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

- **Acceptance testing**
  - Testing with customer data to check that it is acceptable
Testing phases

- Requirements specification
- System specification
- System design
- Detailed design
- Sub-system integration test plan
- System integration test plan
- Acceptance test plan
- Module and unit code and test
- Acceptance test
- Service
- System integration test
- Sub-system integration test
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
Automated process support
CASE

- Computer-aided software engineering (CASE) is software to support software development and evolution processes

- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated translators to generate new versions of a program
Case technology

- Case technology has led to significant improvements in the software process though not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automatable
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these
CASE classification

- Classification helps us understand the different types of CASE tools and their support for process activities

- Functional perspective
  - Tools are classified according to their specific function

- Process perspective
  - Tools are classified according to process activities that are supported

- Integration perspective
  - Tools are classified according to their organisation into integrated units
## Functional tool classification

<table>
<thead>
<tr>
<th>Tool type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning tools</td>
<td>PERT tools, estimation tools, spreadsheets</td>
</tr>
<tr>
<td>Editing tools</td>
<td>Text editors, diagram editors, word processors</td>
</tr>
<tr>
<td>Change management tools</td>
<td>Requirements traceability tools, change control systems</td>
</tr>
<tr>
<td>Configuration management tools</td>
<td>Version management systems, system building tools</td>
</tr>
<tr>
<td>Prototyping tools</td>
<td>Very high-level languages, user interface generators</td>
</tr>
<tr>
<td>Method-support tools</td>
<td>Design editors, data dictionaries, code generators</td>
</tr>
<tr>
<td>Language-processing tools</td>
<td>Compilers, interpreters</td>
</tr>
<tr>
<td>Program analysis tools</td>
<td>Cross reference generators, static analysers, dynamic analysers</td>
</tr>
<tr>
<td>Testing tools</td>
<td>Test data generators, file comparators</td>
</tr>
<tr>
<td>Debugging tools</td>
<td>Interactive debugging systems</td>
</tr>
<tr>
<td>Documentation tools</td>
<td>Page layout programs, image editors</td>
</tr>
<tr>
<td>Re-engineering tools</td>
<td>Cross-reference systems, program restructuring systems</td>
</tr>
</tbody>
</table>
### Activity-based classification

<table>
<thead>
<tr>
<th>Tools</th>
<th>Specification</th>
<th>Design</th>
<th>Implementation</th>
<th>Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reengineering tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debugging tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program analysis tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language-processing tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method support tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration management tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change management tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planning tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Activity-based classification
- Specification
- Design
- Implementation
- Verification and Validation
CASE integration

- **Tools**
  - Support individual process tasks such as design consistency checking, text editing, etc.

- **Workbenches**
  - Support a process phase such as specification or design,
    Normally include a number of integrated tools

- **Environments**
  - Support all or a substantial part of an entire software process.
    Normally include several integrated workbenches
Tools, workbenches, environments

- **CASE technology**
  - Tools
    - Editors
    - Compilers
    - File comparators
  - Workbenches
    - Integrated environments
    - Process-centred environments
  - Environments
    - Analysis and design
    - Programming
    - Testing
      - Multi-method workbenches
      - Single-method workbenches
      - General-purpose workbenches
      - Language-specific workbenches
Key points

- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model.
- General activities are specification, design and implementation, validation and evolution.
- Generic process models describe the organisation of software processes.
- 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
- CASE technology supports software process activities