Software Engineering

Software Development Process Models

Lecturer: Giuseppe Santucci
Summary

• Modeling the Software Process
  – Generic Software Process Models
  – Waterfall model
  – Process Iteration
  – Incremental model
  – Reuse
  – Formal model
  – Extreme Programming
  – Spiral model

• Process activities
To model the software process

• Software products are not **tangible**
• In order to manage a software project the project manager needs **special** methods
• The **monitoring** is based on the explicit definition of **activities** to be performed and **documents** to be produced
  – This means to model the process
• Produced **documents** allow to **monitor** the **evolution** of the process and provide means to evaluate its quality
Process documents

• Software Development Process Models, and their instances, differ from each other for the foreseen activities and for the documents that are produced

• Management likes documentation, while developers hate it:
  – Both are wrong views

• The good compromise
  – Produce documentation useful for the project
The software process

• A structured set of activities required to develop a software system
  – Specification
  – Design and Coding
  – Validation
  – Evolution / Maintenance

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

• The waterfall model
  – Separate and distinct phases of specification and development.

• Evolutionary development
  – Specification, development and validation are interleaved.

• Component-based software engineering
  – The system (part of) is assembled from existing components.

• Formal models
  – Requirements are expressed in a formal language

• There are many variants of these models
  – e.g., formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.
Waterfall process

Analysis

Design

Implementation and unit test

Integration and system test

Maintenance
Waterfall model phases

• Requirements analysis and definition
• System and software design
• Implementation and unit testing
• Integration and system testing
• Operation and maintenance
• One phase has to be complete before moving onto the next phase
Waterfall model’s problems

• One of the main drawback of the waterfall model is the difficulty of accommodating changes after the process is underway
• Iterations are always around
• Initial uncertainty: the end user has not a clear vision of the overall system
• Customer has to wait till a working version of the system is available discovering an error in this phase is dangerous
• Programmers have to wait the analysis phase
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 (Prototypical model/Incremental model);
  – Spiral development.
Prototypal model

- Working prototype: implements only some basic functions (user interface)
- Main objective: collecting and disambiguating requirements involving the customer
- SW tools (e.g., balsamiq mockups)
Prototypal model (example)

- One example from a European project (Promise)

- [Wizard Promise.pptx](Wizard_Promise.pptx)
Incremental model

Similar to the prototypal model but
- Intermediate versions are full working
- It allows for a more accurate design
Incremental development

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

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

System incomplete
Incremental development and 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 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
Formal methods

• Logic or algebraic formalisms for requirement specification, development, and test

• They do not use the natural language (ambiguous)

• Some formal languages do exist (e.g., `Z’, ‘Z++’, ...)

• An example (with an invented syntax)
void Sort(int a[], int n)
// it sorts the array in ascending order

pre: \( \forall \ i \ \forall \ J \ (i \geq 0 \land i \leq n-1 \land J \geq 0 \land J \leq n-1 \land i \neq J) \rightarrow a[i] \neq a[J] \)

post: \( (\forall \ i \ (i > 0 \land i \leq n-1) \rightarrow a[i] > a[i-1]) \land \)
//1. output array elements are sorted

\( (\forall \ K \ (K \geq 0 \land K \leq n-1) \rightarrow \exists \ L \ (L \geq 0 \land L \leq n-1 \land a[K] = \text{old}(a[L])) \) \land
//2. elements in the output belong to the input

\( (\forall \ R \ (R \geq 0 \land R \leq n-1) \rightarrow \exists \ S \ (S \geq 0 \land S \leq n-1 \land \text{old}(a[R]) = a[S]) \)
//3. elements in the input belong to the output
Extreme programming

• An approach 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.
• We will see this process model in detail
Spiral development

• Process is represented as a spiral (rather than as a sequence) of activities
• 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
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.
Core process activities
Process activities

• Let us quickly review the main involved activities
  – Software specification
  – Software design and implementation
  – Software validation
  – Software evolution
Software specification

• The process of establishing what services are required and the constraints on the system’s operation and development
  – Quality requirements!

• Requirements engineering process
  – Feasibility study
  – Requirements elicitation and analysis
  – Requirements specification
  – Requirements validation
The requirements engineering process

- Feasibility study
- Requirement elicitation and analysis
- Requirement specification
- Requirement validation
- Feasibility report
- System model
- User and system requirements
- Requirement document
Software design and implementation

• The process of converting the system specification into an executable system

• Software design
  – Design a software structure that realise 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
  - Sw interfaces
  - User interfaces
- Component design
- Data structure design
- Algorithm design
The software design process

Design activities

- Requirement specification
  - Architectural design
    - System architecture
  - Abstract specification
    - Software specification
  - Interface (UI, SWI) design
    - Interface specification
  - Component design
    - Component specification
  - Data structure design
    - Data structure specification
  - Algorithm design
    - Algorithm specification

Design products
Structured design

• Systematic approaches to software design
• The design is usually documented as a set of graphical models
• Possible models (UML encompasses most of them)
  – Object model
  – Sequence model
  – State transition model
  – Structural model
  – Data-flow model
Programming and debugging

• Translating a design into a program and removing (as much as possible) 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 the program
  – For the error fix
  – For new errors introduced by the fix (regression test)
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
The testing process

- Component testing
- System testing
- Acceptance testing

Integration test
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
  – Integration test

• Acceptance testing
  – Testing with customer data to check that the system meets the customer’s needs
Testing phases

Black box tests MUST be designed NOW!

Requirement specification
  --------------------------
  System specification
  --------------------------
  System design
  --------------------------
  Detailed design
  --------------------------
  Acceptance test plan
  System integration test plan
  Subsystem integration test plan
  Module and unit code and test
  --------------------------
  Service
  Acceptance test
  System integration test
  Subsystem 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. Requirements
2. Assessing existing systems
3. Propose system changes
4. Modify systems

Existing systems
New systems