DAVC19 '06

Software Engineering

Tim Heyer

Karlstad University

Outline (A)

Administrative & Introduction
- Staff
- Structure
- Web

Introduction
- Horror stories
- Software Engineering
- History

Keywords

Staff

- Teachers:
  - Tim Heyer (5A427, 700 2030)
- Course secretary:
  - Inger Bran (5A415, 700 1970)

Course Structure

1. Basic facts (1 pt, level 1)
   - Lectures and reading the course literature (Beck & Scott)
   - A short written exam
2. UP and inspection exercise (1 pt, level 3)
   - Extending use-case, analysis and design models
   - Individual inspection and inspection meeting
3. Reasoning about software engineering issue (3 pts, level 6)
   - Justifying and criticising given statements
   - Two reports

Written Exam

- 10 questions: 4 software engineering, processes and verification & validation, 3 XP and 3 UP.
- All questions are of the type: *"What is (according to the course material and literature respectively) meant by the term X?"*
- Answer should be around 4 sentences (never more than 8).
- Each question gives 0, 1, or 2 points.
- 18 points gives a 5, 16 points a 4, and 14 points a 3.

More Information

`http://www.cs.kau.se/cs/education/courses/davc19/`
or Studenttorget

Mariner I (Venus Explorer)

- 1962-07-22, Cape Canaveral/Florida
- Carrier rocket Fortran control program contained:

```
DO 5 K = 1. 3
...
5 CONTINUE
```

- Carrier rocket leaves flight path and destructs after 290 s
- Costs $ 18,500,000
- Dot instead of comma `DO5K = 1.3`
- Variable declarations not required
- No structured loops
- Blanks in names and numbers allowed

Ariane 5 (Carrier for Satellites)

- 1996-06-04, Kourou/French Guyana
- 30 s after liftoff ground speed reaches a value 5× higher than Ariane 4
- Unprotected float–to–integer conversion results in overflow
- Primary and secondary navigation computers shut down
- Main computer interprets diagnosis data as flight data
- Main computer sends stupid commands to thrusters
- Rocket self-destructs causing 840 million Euro damage and 2–3 years without profits
- The erroneous part was only active after liftoff to allow faster restart in case liftoff was aborted
- No overflow check because it was proven that overflow could not occur *with Ariane 4*
- Only hardware errors were expected, thus identical software on both navigation computers

Mars Climate Orbiter

- Mars Climate Orbiter is feared to have burned up in the Martian atmosphere
- Cost $ 125,000,000
- Spacecraft and navigation teams were using different measurements units
- One team was using Imperial or English measurements (inches, feet and pounds)
- The other team was using metric (centimetres, meters, and kilograms)
- The spacecraft is believed to have passed only 57 km above the surface of the planet instead of the intended 140 km

Therac-25

- Therapeutic linear accelerator with beams of x-rays or electrons
- 20.000 lines of code by single programmer over several years
- People died in the 80th because of overradiation due to software errors
- Console indicated no or to low dose administered
- Extremely poor coding style
- Showing correctness and re-creation of errors was extremely hard
- One of the errors only occurs when radiation type was changed late
- Good software engineering is important
- Other means are recommended though (fuse)

AT&T telephone system

- 1990
- 70 millions long distance calls out of 138 millions could not be served under 9 hours
- Costs $ 75 millions at AT&T and several $ 100 millions at customers
- Reason was software error

How Software Development Projects Fail

- No functioning software results
- The resulting software does not adequately address the need of the users
- Software contains incorrect computations
- The software is too difficult to use correctly
- The system response time is too slow to be used without frustration

Software Engineering

IEEE Standard 610.12:

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
2. The study of approaches as in 1.

Why Software Engineering is not Universal

- Understanding software development as programming only without recognition of importance of analysis and design
- Short-sighted technical management
- Poor project estimation, thus unreasonable deadlines

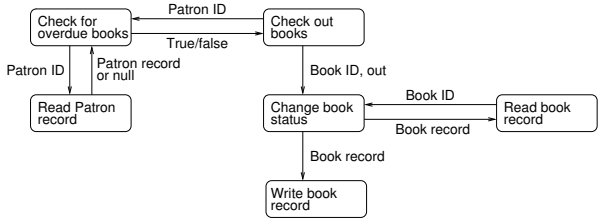
A Brief History of Software Engineering Techniques

- Structured Programming
  - No gotos
- Functional Decomposition
  - Top-down organisation of subprograms
- Structured Analysis
  - Recognition that analysing the problem statement has critical influence on the success of the overall project
  - Formal modelling of subprogram interaction with dataflow diagrams

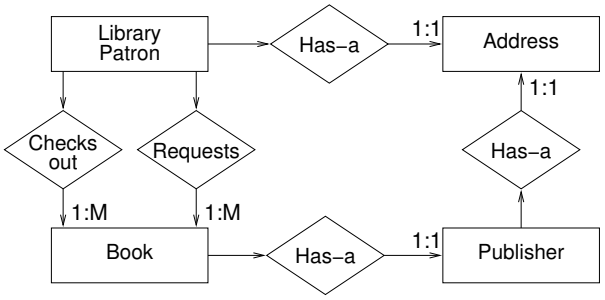
More History of SE Techniques

- Data-centered analysis
  - Uses techniques developed in structured analysis
  - Data modelling occurs using entity relationship diagrams before functional modelling
- Object-oriented analysis
  - No longer segregates the modelling of functions and data
  - Objects aggregate data with functions that operate on the data

Sample Dataflow Diagram



Sample Entity Relationship Diagram



Keywords

Software engineering.

Outline (B)

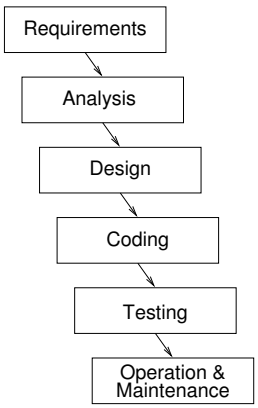
Process

- To provide a service or to develop a product a sequence of tasks is performed
- The set of ordered tasks can be considered a process
- A process usually involves tools and techniques
- The organisation and discipline in the activities are acknowledged to contribute to the quality and shorter development time
- Processes that involve the creation of a product are also referred to as *life-cycles*

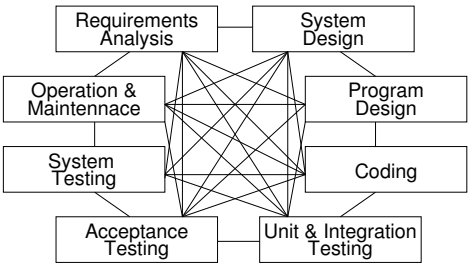
Software Process Models

- A software process model describes how the development of software should progress respectively progresses
- Process models create a common understanding what should be done
- Process models help finding inconsistencies, redundancies etc. in the process
- Many software process models have been proposed

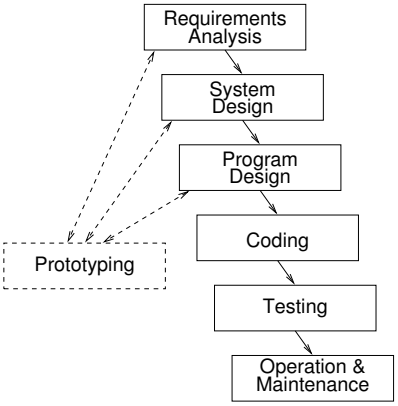
Waterfall Model



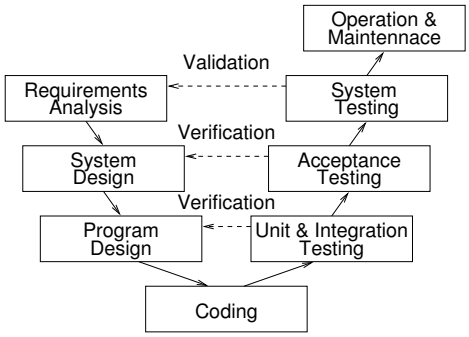
Software Development Process in Reality



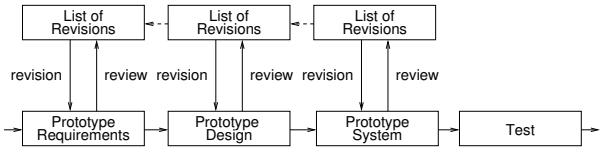
Waterfall Model with Prototyping



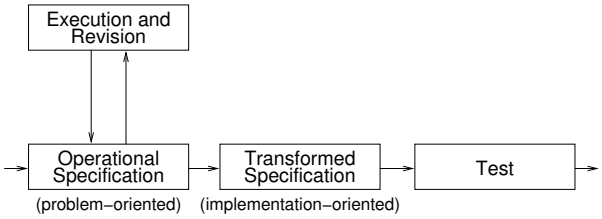
V Model



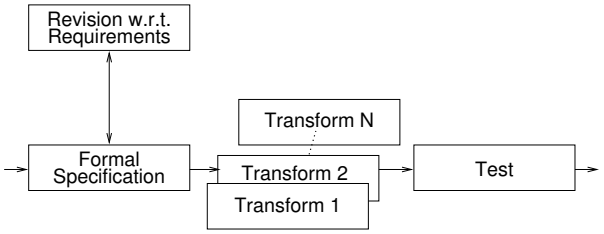
Prototyping Model



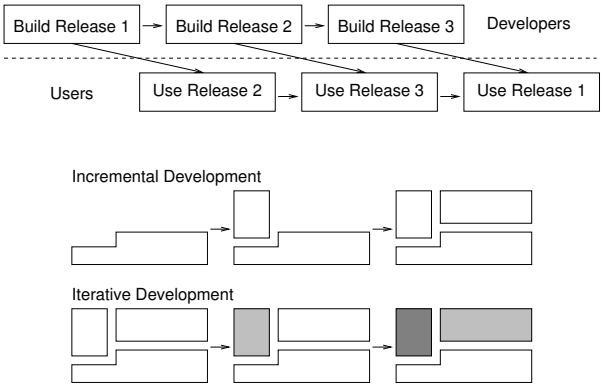
Operational Specification



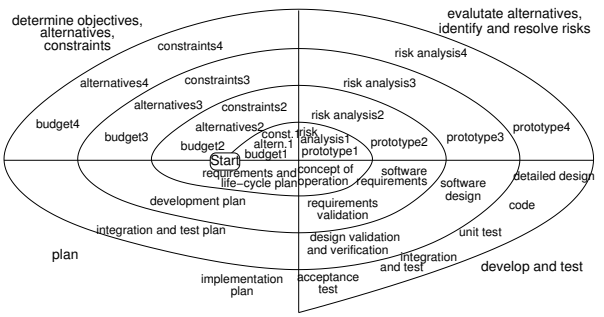
Transformational Model



Incremental/Iterative Development



Spiral Model



Some Remarks

- Many more process models than the ones presented here are in use
- As shown many activities are common to all process models
- The focus is on technical aspects of software development
- Behavioural and organisational aspects are captured only to a small extent
- Orthogonal model covering the latter aspects exist

Keywords

Process, life cycle model, maintenance, waterfall model, incremental development, iterative development.

Outline (C)

What Is XP?

Beck, first edition:

XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.

Beck, second edition:

XP is giving up old, ineffective technical and social habits in favor of new ones that work; XP is fully appreciating yourself for total effort today; XP is striving to do better tomorrow; XP is evaluating yourself by your contribution to the team's shared goals; XP is asking to get some of your human needs met through software development.

Why Is It Named "eXtreme"?

- Reviews are good, thus code is reviewed all the time
- Testing is good, thus everybody tests all the time
- Design is good, thus everybody designs all the time
- Simplicity is good, thus design is kept as simple as possible to support the current functionality
- Architecture is important, thus everybody defines and refines the architecture all the time
- Integration testing is important, thus integration and testing occurs several times a day
- Short iterations are good, thus iterations are very short

Four Control Variables of Software Development

Cost	Time	Quality	Scope

- External forces pick the values of any three of the variables
- Development team picks the value of the fourth variable
- External forces can not pick the values of all four variables

More on Scope

- Scope is the most important variable of software development
- Managing scope gives managers and customers control over cost, quality, and time
- Requirements are never clear at first but change through experience
- Scope is very soft enabling shaping it more easily
- Cost, time, and quality can be kept by continually adjust scope
- A software development discipline based on this model would have to tolerate change easily

Metaphor: Learning to Drive

Beck's mom:

Driving is not about getting the car going the in the right direction. Driving is about constantly paying attention, making a little correction this way, a little correction that way.

Values, Principles, and Practices

Practices	Things that are actually done
Values	Values determine what you do like and do not like
Principles	Domain-specific guidlines for life

The Values of XP

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Principles

- Humanity
- Economics
- Mutual benefit
- Self-similarity
- Improvement
- Diversity
- Reflection
- Flow
- Opportunity
- Redundancy
- Failure
- Quality
- Baby steps
- Accepted responsibility

Primary Practices

- Sit together
- Whole team
- Informative workspace
- Energized work
- Pair programming
- Stories
- Weekly cycle
- Quarterly cycle
- Slack
- Ten-minute build
- Continuous integration
- Test-first programming
- Incremental design

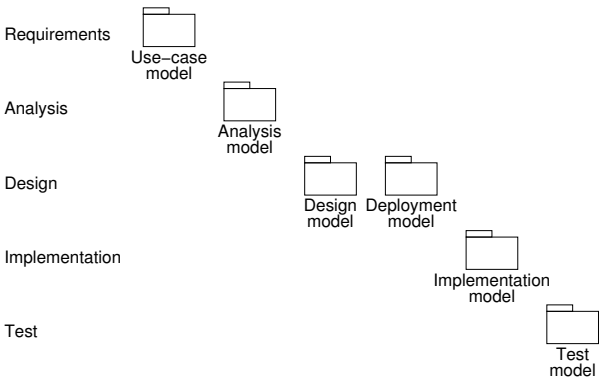
Risk in XP

Risk	**XP treatment**
Schedule slips	Short release cycles
Project canceled	Customer involvement
System goes sour	Comprehensive suite of Test cases
Defect rate	White- and black-box testing
Business misunderstood	Customer involvement
Business changes	Short release cycles
False feature rich	Important tasks are implemented first
Staff turnover	Higher programmer responsibility

What is the UP?

- The UP is a generic process *framework* for the development of software
- Distinguishing aspects are that the UP is *use-case driven*, *architecture-centric*, and *iterative and incremental*
- The UP uses the UML

The UP is Use-Case Driven



Why Use-Cases?

- To capture the value adding requirements
- To drive the process
- To devise the architecture and more

The UP is Architecture-Centric

- The architecture can be considered the common vision that all workers must agree on or at least accept
- The UP is architecture-centric because it involves using the architecture as the key to conceptualizing, constructing, managing, and evolving the system being built

Why do we need an architecture?

- To understand the system
- To organize development
- To foster reuse
- To evolve the system

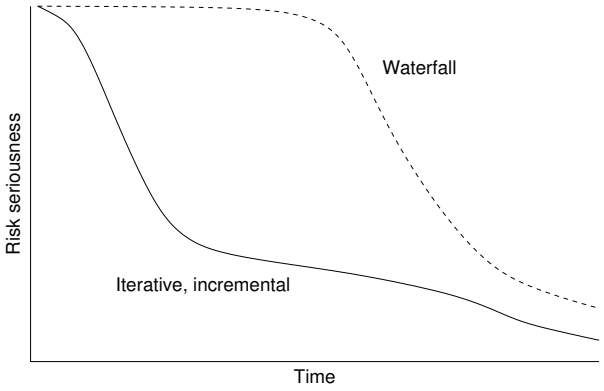
The UP is Iterative and Incremental

- The strategy is to developing a software product in small manageable steps, i.e. you plan, specify, design, implement, integrate, and test a little

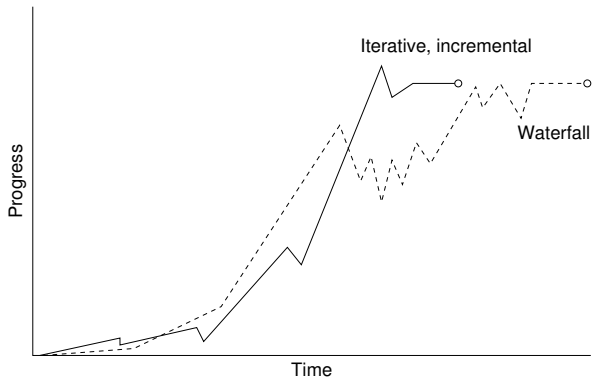
Why iterative and incremental?

- To mitigate risks
- To get a robust architecture
- To handle changing requirements
- To allow for tactical changes
- To achieve continuous integration
- To attain early learning

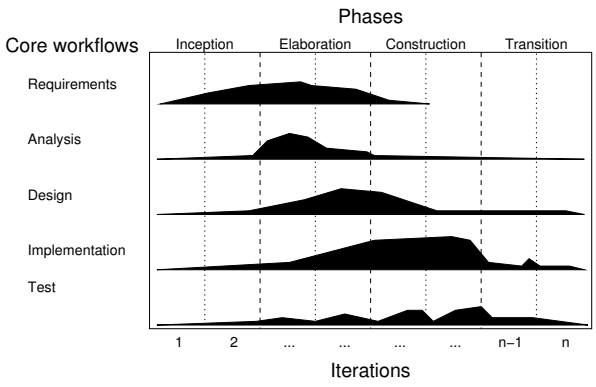
Mitigating Risks



Achieving Continuous Integration



Iterations over the Life-Cycle



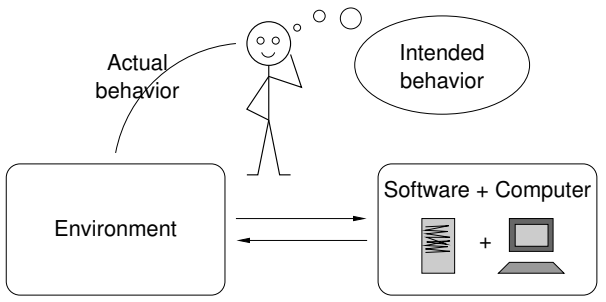
Keywords

Practice, value, principle, communication, simplicity, feedback, courage, humanity, mutual benefit, self similarity, failure, baby steps, accepted responsibility, sit together, whole team, informative workspace, pair programming, stories, weekly cycle, test-first programming.

Use-case driven, architecture-centric, iterative and incremental, inception phase, elaboration phase, construction phase, transition phase, requirement workflow, analysis workflow, design workflow, implementation workflow, test workflow, use-case model, analysis model, design model, deployment model, implementation model, test model.

Outline (D)

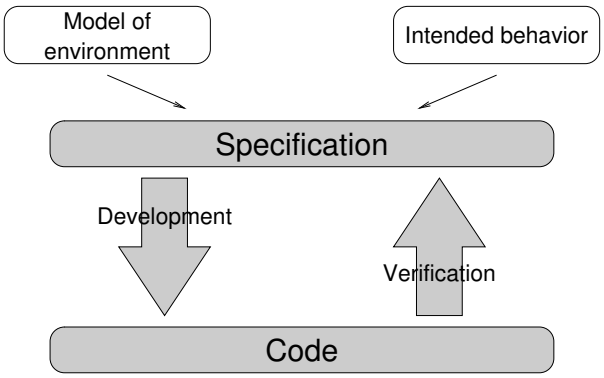
What Is a Major Goal of Software Development?



Goal

Actual behavior = intended behavior (correctness)

Components of Software Development



What Is Testing and Debugging?

Testing is the process of determining the existence of a fault (typically by executing a piece of software)

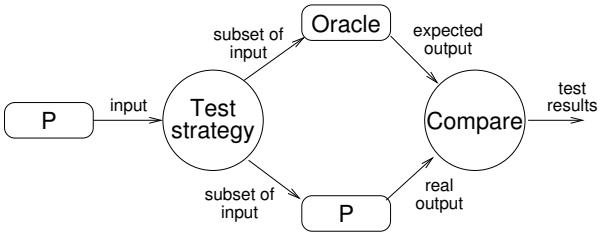
Debugging is the process of finding and correcting a known fault

Failure is the inability of a piece of software to perform according to its specifications

Fault is a manifestation of an error in the software

Error refers to some human action that results in a fault in the software

Global View of the Testing Process



Classification According to Adequacy Measurement

Coverage-based testing: Testing requirements based on coverage of the artifact to be tested, e.g. certain amount of statements, branches, paths

Fault-based testing: Testing requirements based on ability to detect faults, e.g. fault seeding, mutation testing

Error-based testing: Testing requirements based on knowledge of typical errors made by people, e.g. off-by-1 errors at boundary values

Classification According to Information Source

Black-box testing: Nothing is known about the internal structure of the code, test cases aim to represent all possible inputs

White-box testing: Knowledge of the programming constructs is used to determine the test cases to use, e.g loops are tested for 0, 1, max, and max + 1 iterations, conditions are tested for true and false

Different Test Stages

Unit testing: The units comprising a system are individually tested

Integration testing: The composition of components is tested

System testing: The whole system is tested against user doc and requirements spec

Acceptance testing: The whole system is tested against customer expectations

Installation testing: If the system has become operational in a different environment

Regression testing: Retesting elements of the system that were tested in a previous version or release

Examples: Testing a Correct Program

Computing n^2 without multiplication:
$1 + 3 + 5 + \cdots + (2n - 1) = n^2$, i.e. crossing out each second integer.

Example: Testing an Incorrect Program

Compare two strings for equality.

Drawbacks of Testing

Testing is a common technique to increase confidence in program correctness, however:

- Very late in software development
- Can only show the presence of bugs, but not their absence

What Is Inspection?

- Inspection is the process of finding defects in the artifact by human examination
- Artifacts can be any written document, i.e. specifications, source code, contracts, test plans, test cases, etc.
- Inspection is usually performed by 3 to 5 participants
- Participants often have different roles or assume different perspectives, e.g. customer, implementer, tester etc.

The Basic Inspection Process

Initiation and Documents

- An author gives artifacts to a moderator and asks for an inspection
- The moderator recruits a team of inspectors and gives them the artifact and all other necessary documents
- A kick-off meeting ensure that all participants understand the artifact to inspect as well as their roles in the inspection

Checking

- Inspectors individually read the artifact and note all the defects found with the help of checklists, rules etc.
- The inspection team conducts a defect logging meeting

Completion

- The author takes the defect log and fixes all the logged defects

Example: Rules

General:

- All documents shall be unambiguous to the intended readership.
- Ideas shall be stated once only in documents and thereafter referred to by their unique tag.

Source Code:

- The code should use symbolic constants, instead of hard-coded values, whenever possible.
- The level of commentary should match the complexity of the code.

Requirements:

- Requirements must be stated in terms of final need, not perceived means.

Advantages and Disadvantages

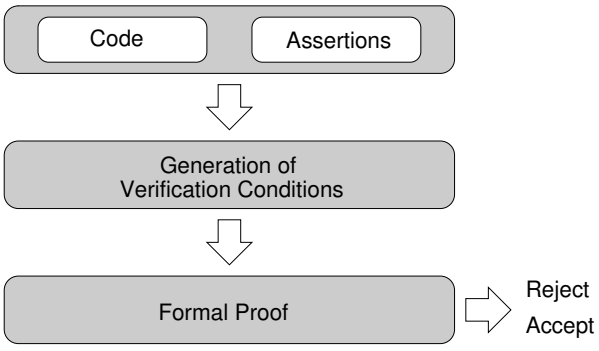
Advantages:

- All types of artifacts can be inspected
- Applicable early in the process
- Effective
- Provides value in improving software reliability, availability, and maintainability

Disadvantages:

- Rather informal
- Result depends much on the discipline and experience of the involved personnel

Hoare's Partial Correctness Assertion Method



Example

Code and assertions:

$$\{X = x \wedge Y = y\} x := x + y; y := x - y; x := x - y \{x = Y \wedge y = X\}$$

Verification condition:

$$x = X \wedge y = Y \Rightarrow (x + y) - ((x + y) - y) = Y \wedge ((x + y) - y) = X$$

- Assertion: condition that program variables must satisfy
- Precondition: assertion describing properties of input
- Postcondition: assertion describing properties of output

Advantages and Disadvantage

Advantages:

- Provides a formal proof of the correctness
- Applicable early in the development
- Supports the development of software artifacts

Disadvantages:

- Required formal rigor is often considered a major drawback

Keywords

Correctness, testing, debugging, failure, fault, error, coverage-based testing, fault-based testing, error-based testing, black box testing, white box testing, unit testing, integration testing, acceptance testing, inspection, formal verification.

Outline (E)

UML
- Introduction
- Modelling
- History
- Diagrams and Views

Assignment
- Scenario
- Deliverables
- Process

What is the Unified Modelling Language (UML)?

- A model is an abstract representation of some other thing (which may be real)
- UML is a standardised language for specifying, visualising, constructing, and documenting different kinds of systems, ranging from software to organisational processes
- UML represents a collection of engineering practices that are used for the modelling of large and complex systems

Why to Model?

"When it comes down to it, the real point of software development is cutting code"

"Diagrams are, after all, just pretty pictures"

"No user is going to thank you for pretty pictures; what a user wants is software that executes"

[M. Fowler: UML Distilled, Addison Wesley, 1997]

We Model Because . . .

Good models are necessary for:

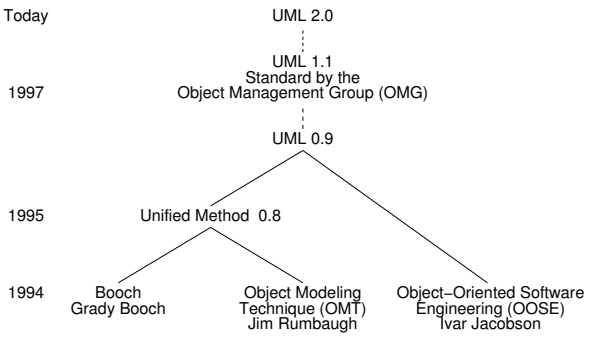
- Making complex systems more understandable
- Visualising the essential aspects of a system
- Communication among project members and with the customer
- Ensuring architectural soundness

A good model is more easily manipulated and understood than the thing it represents

What the UML Is *Not*

- It is not a method or a process, i.e. UML does not specify *how* to model a system.
- It is not a tool.
- It is not a programming language, but a visual modeling language.

UML History



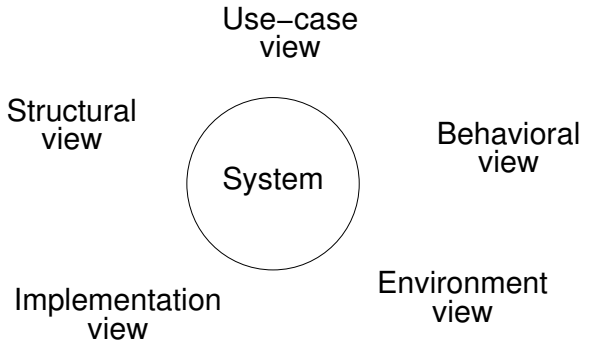
UML Diagram Types

- Use-case diagrams
  For modeling the functionality provided by a system
- Sequence diagrams
  For modeling interactions within a system (focusing on timing)
- Collaboration diagrams
  For modeling interactions within a system (focusing on the structural organisation of the objects)
- State diagrams
  For modeling the behavior of system objects
- Activity diagrams
  For modeling the behavior of use-cases, objects, and operations

UML Diagram Types (cont'd)

- Class diagrams
  For modeling the static structure of classes
- Object diagrams
  For modeling the static structure of objects
- Component diagrams
  For modeling components
- Deployment diagrams
  For modeling productive deployment of a system

UML Views



Scenario

You recently got employed by a company that develops a document version control system for the Unified Process. Key elements of the tool are:

- The artifacts are stored in a central repository. The artifacts get version numbers.
- Users log in to check in and/or check out artifacts from the central repository.
- The system respects the users' roles and the current workflow.

Your Task

Specify, analyse, and design two use-cases:

- *Check In*: enables a worker to check in an artifact.
- *End Activity*; enables a worker to end a workflow (i.e., an activity) and to start the next subsequent workflow.

Remarks:

- You should carefully consider when and who may check in which artifacts and end which workflow.
- You may use whatever tools you want to create documents and diagrams. For the diagrams we actually suggest pencil and paper.

Extend the use-case model

- A use-case diagram containing the existing and new actors and use-cases.
- Two use-case descriptions, each consisting of a brief description, an activity diagram describing the flow of events, a precondition, a postcondition, and special requirements.

Extend the analysis model

- An analysis class diagram containing the existing and new classes.
- Two analysis use-case realizations (one for each of the above use-cases). Each use-case realization is supposed to consist of a collaboration diagram, a description of the event flow, and special requirements.

Extend the design model

- A design class diagram for the server package containing the existing and new classes (omit existing attributes and operations).
- Two design use-case realizations (one for each of the above use-cases). Each use-case realization is supposed to consist of a sequence diagram.
- A document briefly describing each new operation.

Lab Process

1. Apply for a pair on the web page.
2. Solve the assignment in that pair; make two copies of your reports.
3. Apply for a inspection pair on the web page.
4. Meet with your inspection pair and exchange your report; check quickly that the reports you are supposed to inspect are acceptable.
5. Produce your individual inspection reports.
6. Sign up for an inspection meeting on the list at the pinboard outside room 5B403.
7. Perform the inspection meeting with the teacher as moderator. Both pairs have to be present at the meeting.

Outline (F)

Report
- Introduction
- Process
- FAQ

References
- Publications
- Searching

Assignment

You are supposed to:
1. justify one of a given set of statements and
2. criticise one statement.

Perspective A

Always

Design documentation, like class and sequence diagrams, should always be created and kept complete and up to date.

Perspective B

Initially

Design documentation, like class and sequence diagrams, should only be created for the initial development of the system but should not be kept up to date later.

Perspective C

Sometimes

Design documentation, like class and sequence diagrams, may only be created when design ideas need to be communicated during development. The documentation should be discarded afterwards.

Perspective D

Never

Design documentation, like class and sequence diagrams, should never be created. Instead code should be used to communicate design ideas during development.

Report Process

1. Choose the perspective that is closest to your own opinion.
2. Justify that perspective (5 references; 400–450 words, excluding references, quotations and appendixes; English).
3. Apply for a partner on the web page.
4. Meet with your partner and exchange your reports.
5. Criticise the other student's justification (5 references; 400–450 words, excluding references, quotations and appendixes; English).
6. Hand in your reports: your own justification, your critique of another student's justification, and a copy of the report you criticised.

Remarks

- Write your own argumentation individually.
- Choose the perspective that is closest to your own opinion.
- References can, e.g. be conference articles, journal papers, white papers, books, web pages. The quality of your references will affect your mark.
- You may include scientific papers that you do not have full access to in your reference list, if 1) the abstracts support your argumentation, and 2) you include the abstracts in your report as an appendix.
- Marks are based on: language, argumentation, references, presentation and structure.

FAQ

- I do not really understand how the reports are supposed to be. Where do I start?
- Can I use the XP book by Beck as a reference?
- The student who's justification I am supposed to criticise picked the same statement as me. What should I do? It is difficult to criticize the other student's justification since I have the same opinion.

Scientific Publications

- Journal article
- Conference article
- Reports
- Dissertations
- Books

Databases

1. Selection
2. Indexing
   Keywords, thesaurus terms et cetera.
3. Integration and distribution

Search Process

1. Preparation
2. Search
3. Results
4. Evaluation