

Object-oriented Analysis and Design: Introduction and Context

Dr Peter Popov

30th of September 2025

Objectives of this Lecture

- [Why Software Engineering](#)
- Software Engineering and Software Process Models
- Difficulties with specifications
- Object-oriented analysis and design
- Brief introduction to UML

Part 1: Software Engineering and Software Process Models

IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 3

3

Why Software Engineering

Software engineering is a “relatively” new discipline, proposed in 1968 to address the **software crisis**. SW projects are:

- Over-budget;
- Over-time;
- Low quality software (inefficient, buggy, difficult to maintain);
- Not meeting requirements;

The crisis was recognised in late 60s

- 1969–1970 NATO Software Engineering Conferences
(<http://homepages.cs.ncl.ac.uk/brian.randell/NATO>).

Software engineering: “an engineering discipline concerned with theories, method and tools for cost-effective development of software systems”

(Ian Sommerville - SE, 9th Edition)

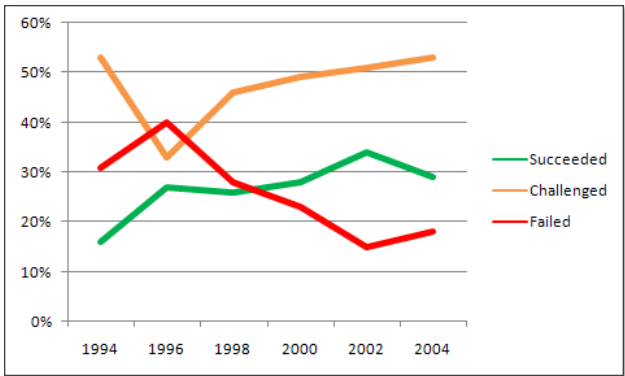
New techniques and methods were necessary to *control the complexity* inherent in large software systems

IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 4

4

Why Should We Care?



For 2019 the “software development CHAOS” continues:

- 31.1% of the IT projects are cancelled.
- 52.7% of projects will cost 189% of their original estimates, etc..

If interested can check the most recent CHAOS report at:

<https://www.successthroughsafe.com/blog-1/2021/11/13/standish-chaos-report-2021>

Would **you** buy a car that only had a 30% chance of driving off the lot with **no** problems?

Source: The Standish Group CHAOS report:

<https://www.projectsmart.co.uk/white-papers/chaos-report.pdf>

Recent Significant IT Failures

Company	Year	Outcome
Hudson Bay (Canada)	2005	Inventory system problems lead to \$33.3 million loss.
UK Inland Revenue	2004/5	\$3.45 billion tax-credit overpayment caused by software errors.
Avis Europe PLC (UK)	2004	Enterprise resource planning (ERP) system cancelled after \$54.5 million spent.
Ford Motor Co.	2004	Purchasing system abandoned after deployment costing approximately \$400 M
Hewlett-Packard Co.	2004	ERP system problems contribute to \$160 million loss.
AT&T Wireless	2004	Customer relations management system upgrade problems lead to \$100M loss

Recent UK example – Terminal 5 luggage handling SW was not tested enough.

Problematic Projects (Yourdon)

Mission Impossible

- Likely to succeed, happy workers

Ugly

- Likely to succeed, unhappy workers

Kamikaze

- Unlikely to succeed, happy workers

Suicide

- Unlikely to succeed, unhappy workers

Who is Ed Yourdon: https://en.wikipedia.org/wiki/Edward_Yourdon

Roadmap

- Why Software Engineering
- [Software Engineering](#)
- Software Process Models
- Difficulties with specifications
- Object-oriented analysis and design
- Brief introduction to UML

What is Engineering?

Engineering is the profession in which a knowledge of the mathematical and natural science, gained by study, experience and practice, is applied with judgement to develop ways to utilise economically the materials and forces of nature for the benefit of mankind.

(Engineers' Council for Professional Development)

What is Software?

Computer **programs** and associated **documentation** necessary to allow programs to operate correctly:

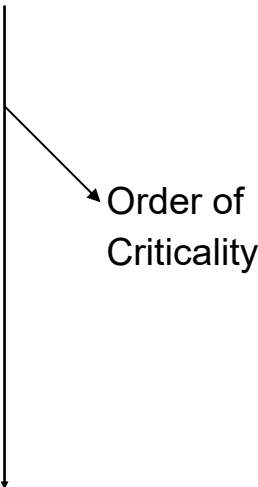
- separate programs
- configuration files to set up the programs
- system documentation - describes the systems structure
- user documentation - explains how to use the system
- web sites - for users to download product information

Software engineers are responsible for developing software products that can be:

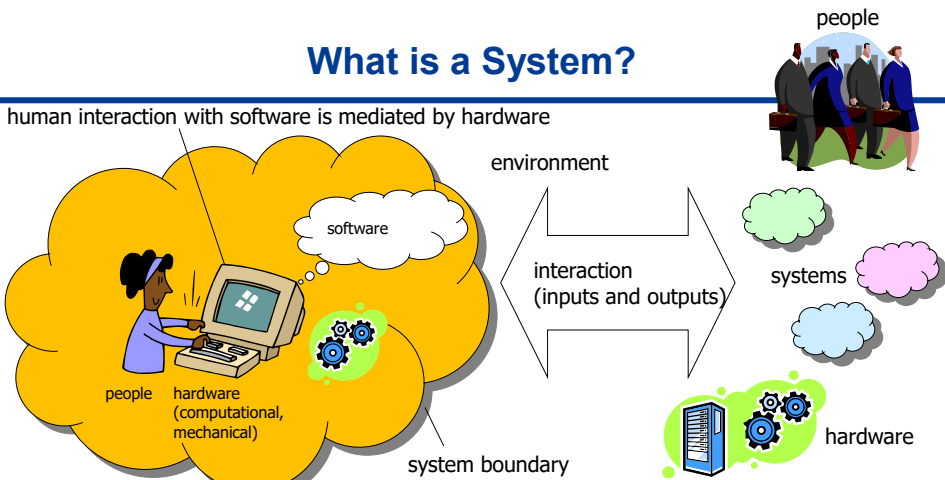
- **Generic**: stand-alone systems which are produced to be sold/given away (including open source) to a range of different customers
- **Bespoke** (customised): systems commissioned by a particular customer according to their specifications

Software Applications (examples...)

- Computer games
- Washing machines
- Word processors
- Operating Systems
- Automated Teller Machines
- Avionics Control
- Nuclear reactor control



What is a System?



- A system is a **bounded** physical/virtual entity consisting of interacting elements operating in an environment to achieve defined objectives
- A system needs to interact with its **external environment** to achieve its objectives
- The choice of system boundary is somewhat arbitrary – it depends on the problem you are trying to solve!
 - What are the system boundaries of the Internet?

System Boundaries

- Every system has *boundaries*
- Defining system boundaries is an important step in system analysis:
 - Boundaries are not always obvious:
 - What are the system boundary of:
 - The Internet?
 - Of the Amazon cloud?
 - Usually, boundaries become clearer when one asks the following questions:
 - What is the purpose and scope of the system
 - What is *outside* the scope of the system
- Inside system boundaries are the system *components*
- Outside system boundaries is the system *environment*

IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 13

13

Emergent System Properties

- Properties of the **system as a whole**. They can only be measured when the subsystems have been integrated to form the complete system
- **Functional properties**: they appear when all parts of a system work together to achieve some objectives
- **Non-functional properties**: they are related to the behaviour of a system and are normally critical for computer-based systems (e.g., reliability, performance, safety, security)

IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 14

14

What is Software Engineering?

- An engineering discipline that is concerned with all aspects of software production ranging from system **specification** to **maintenance**
- Software engineers should:
 - adopt a **systematic, disciplined** and **quantifiable** approach to the **development, operation, and maintenance**,
 - use appropriate techniques and tools depending on:
 - the problem to be solved,
 - the development constraints and
 - available **resources**
 - Need to produce something that is **good enough**, given the **time and resource limitations**
 - Perfection (i.e., software that never fails/free from bugs) is an aspiration but is **rarely feasible**!

SW Body of Knowledge

<https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3>

Why is Software Engineering hard

The computer revolution

- General purpose machine + software = Special purpose machine
 - Mobile apps for measurements of calories, steps, IoT, Smart home, etc.

"Curse of flexibility"

- Software design can be **changed** easily (no retooling needed).
- Software is so flexible that one can start working with it before fully understanding what is needed
 - "Software is the resting place of **afterthoughts**."
- The untrained can get **partial success**. "Scaling up is hard to do".

And they looked upon the software and saw that it was good. But they just had to add one other feature ..."

Organized complexity (Weaver)

- The system consists of a very **large number of interacting parts**
- **Probabilities and statistics** are adequate for describing the system behaviour, but rarely used in software engineering

What is complexity

- The underlying factor is *intellectual manageability*
- A "simple" system has a **small number of unknowns** in its interactions within the system and with its environment.
- A system becomes intellectually unmanageable when the level of interactions reaches the point where they cannot be **thoroughly**:
 - Planned
 - Understood
 - Anticipated
 - Guarded against
- In this module we will practice with UML models, which allow us to start with the simple and add complexity (gradually) later
 - If you start with implementation, then the scope for increasing complexity gradually are quite limited.

SW Engineers

- Software Engineer = (Master) Programmer?
- Civil Engineer = (Master) Builder?
- Artisans can build products (sometimes amazing ones!).
- Engineers can build products following **a process** and **demonstrate** that these have specific properties, such as.
 - Correctness
 - Reliability
 - Performance
- If you cannot demonstrate that your artifact has **specific properties** (e.g., **reliability or security**), then you haven't engineered it.
- Sometimes artisans can build products that engineers cannot demonstrate that they have specific properties.
 - No warranties...
- Genuine difficulties **demonstrating** ultra-high software reliability.

Software Engineering & Computer Science

- Computer science is concerned with the **theories** and **methods** that underlie computers and software systems.
- Software engineering is concerned with the **practical problems** of *producing* software.
- **Knowledge** of computer science is essential for software engineers.

David Parnas discusses the difference between Computer Science and Software Engineering in:

D. L. Parnas, "Education for computing professionals," in Computer, vol. 23, no. 1, pp. 17-22, Jan. 1990, doi: 10.1109/2.48796.

Professional and Ethical Responsibility

- The job of software engineers involves large responsibilities **beyond the application of technical skills**
- Software engineers must behave in an **honest and ethical way**
- Standards of acceptable behaviour are not bounded by laws, but by other notion of professional responsibility:
 - Confidentiality
 - Competence
 - Intellectual property rights
 - Computer misuse
- ACM, IEEE and British Computer Society have published a code of professional conduct (code of ethics)
 - see Sommerville's textbook for extensive discussion!

Takeaway Messages (Part 1)

- Software Engineering solves a real need of making software development similar to other engineering disciplines
- Software Engineering evolved, but software disasters are still common due to the **sheer complexity** of the software systems being developed.

Further Reading (Part 1)

- The material in this lecture is **not covered** in the recommended textbook. There are numerous sources (including on-line) that can be used.
- I would recommend:
 - Ian Sommerville's "Software Engineering", Edition 6 - 9: Chapter 1, 2.
 - The University library should have a significant number of copies.

Part 2: Software Process Models

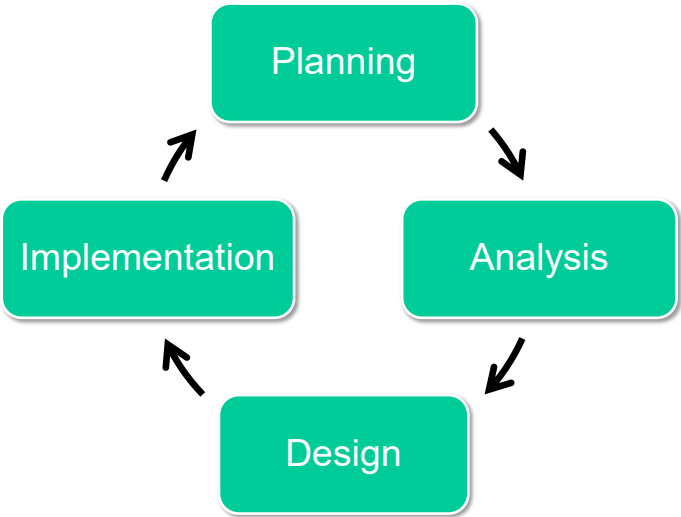
Roadmap

- Why Software Engineering
- Software Engineering
- **Software Process Models**
- Software Requirements Engineering
- Object-oriented analysis and design
- Brief introduction to UML

What is Software Process?

- Set of **activities** and **results** which produce a software product (also known as **software development life cycle, SDLC**)
 - **specification**: definition of the functionality of the software and of the development constraints
 - **development**: production of the software system (i.e., design and implementation)
 - **validation**: assurance that the software does what the customers required (typically testing, but may involve some other forms of assurance, e.g., Proof of correctness, etc.)
 - **evolution**: changing the software to comply to changing demands
- The time and results of the activities vary
- Different processes may be used to produce the same type of product
 - The choice of a particular process is affected by the “organisational culture” or personal preferences.
- A term related to SDLC is software product lifecycle (S)PLC.
 - It extends beyond SDLC and includes software **maintenance** (i.e., release of new versions, patching, etc.) and eventually product **obsolesce**.

Software Development Life Cycle (SDLC)



SDLC: Planning

1. Project Initiation
 - Develop a system/software request
 - Conduct a feasibility analysis
2. Project Management
 - Develop work plan
 - Staff the project
 - Control and direct the project

Why should we build this system?

SDLC: Analysis

1. Develop an analysis strategy
2. Gather and engineer requirements
 - The requirements by the different stakeholders may differ (even be contradictory).
 - An important part of the *analysis* is to identify the problems and resolve them one way or another
3. Develop a system *proposal* and a specification

What should the system do for us?

Where and when will it be used?

SDLC: Design

1. Develop a design strategy
2. (High Level Design) Design **architecture** and **interfaces** (between the subsystems and its environment, e.g., user interface)
 - shows the sub-systems (components) and how they interact with one another, but does not show details about the implementation of the components (a powerful abstraction)
 - Some components may be acquired **off-the-self**, i.e., use (commercial) off-the-shelf software
3. Develop **databases** and **file** specifications
4. (Low level design) Develop the program design (i.e., **algorithms**)

How will we build the system?

SDLC: Implementation

1. Construct system
2. Install system
 - Implement a training plan for the users
3. Establish a support plan (maintenance, training, etc.)

Build the system!

Putting the SDLC Together

- Each phase consists of steps that lead to specific deliverables
- The system evolves through *gradual refinement*
- Once the system is implemented, it may go back into a planning phase for its next revision, a follow-on system, or maintenance releases

Software Engineering Process

Purpose:

- to allow managers and software engineers to understand and control the software development activity

Defines:

- *Who* – the roles involved in the software development activity
- *What* – the activities the roles need to perform to develop the software
- *When* – the sequencing of activities

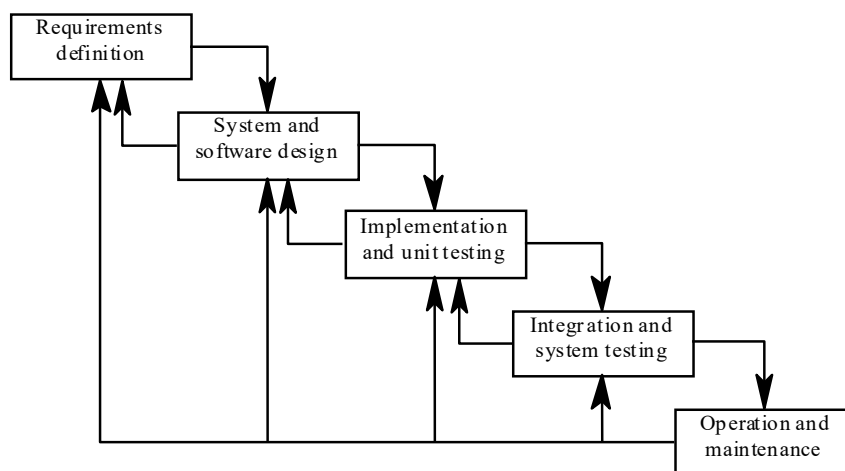
Software Process Models

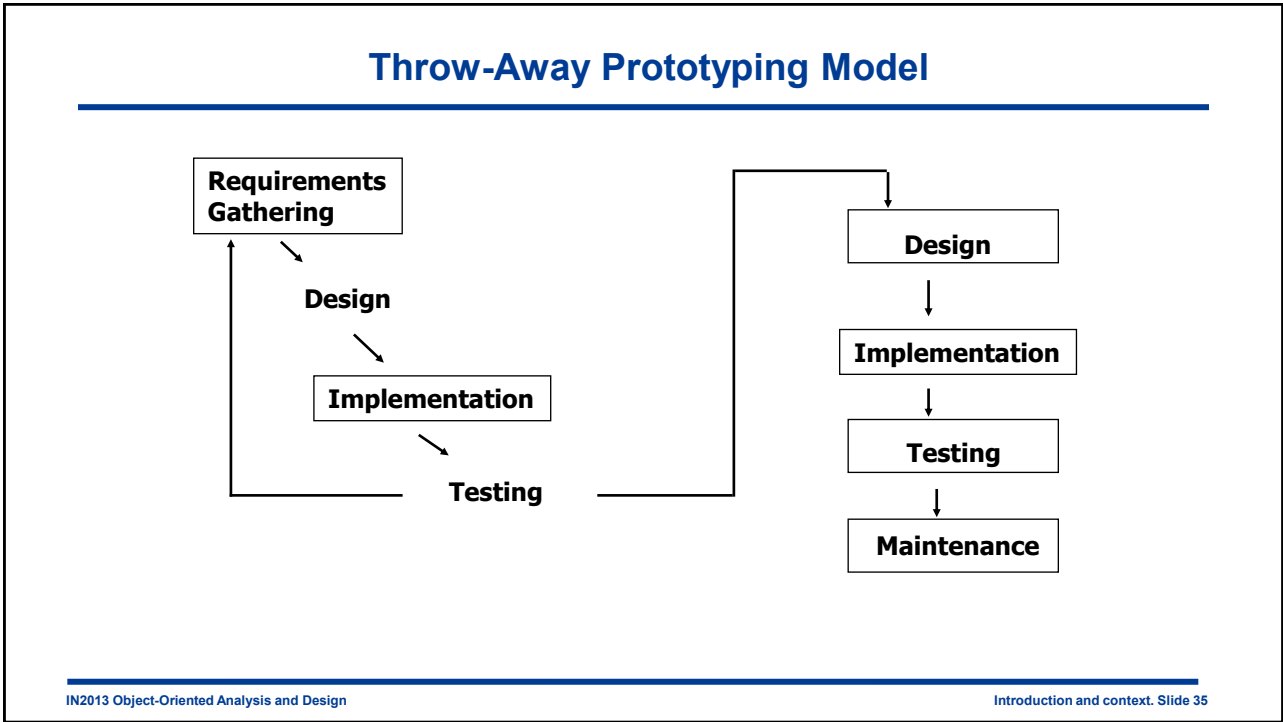
A *simplified description* of the engineering process presented from a particular perspective (an abstraction of the process)

Some well-known *examples* of software *process models*:

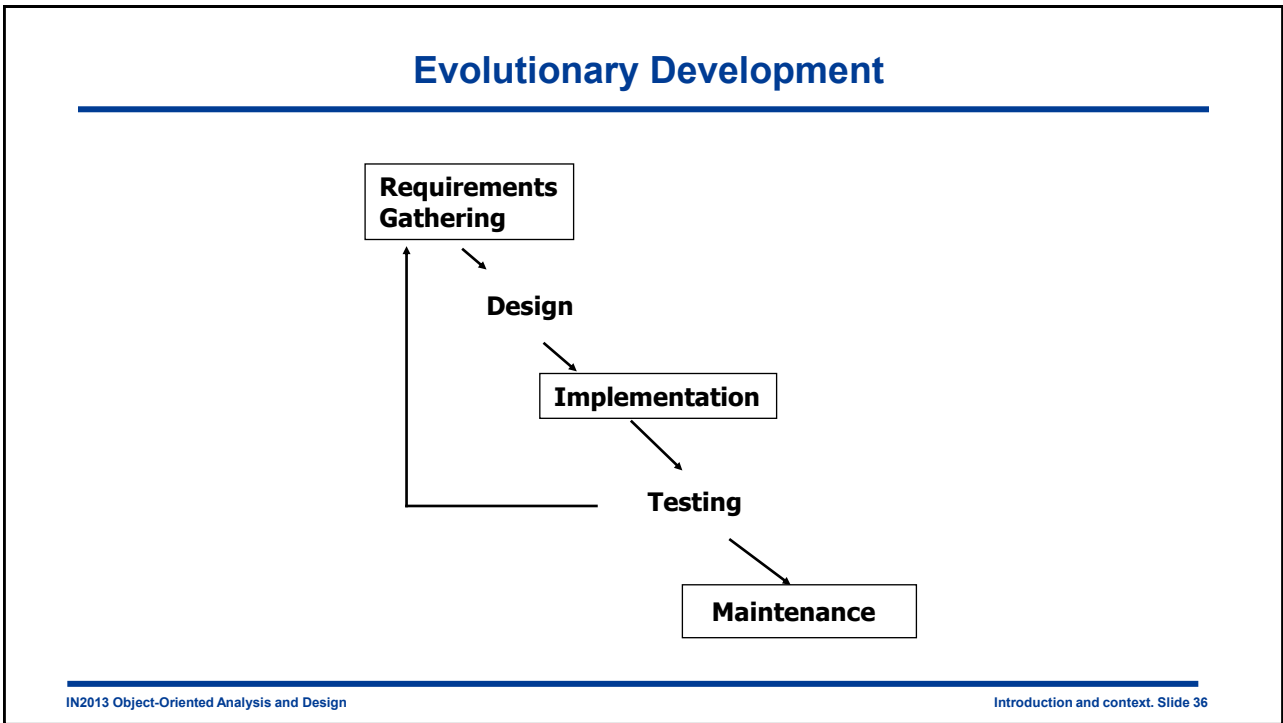
- Waterfall
 - Separate and distinct phases of specification and development. The entire system must be specified **before** the design can start, etc. Very rigid. Rarely used in its pure form.
- Prototyping
 - Use prototypes to elicit requirements and/or validate specifications. Useful in case the customer is unclear about what they really want.
- Evolutionary development
 - Specification and development are interleaved

Waterfall Model





35



36

Other process models

- V-model
- Spiral model,
- Unified process (UP)
 - Usually linked with UML as UP and UML were defined simultaneously (and by the same people who developed the initial version of UML)!
 - UML can be used with any process model.
- Agile methods
 - eXtreme Programming
 - SCRUM,
 - etc.
- In practice, a software engineering process includes elements of different process models.

I leave learning about these models for independent **self-study**.

Agile manifesto

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan.

That is, while there is value in the items on the right, ***we value the items on the left more.***”

(<http://agilemanifesto.org/>)

Takeaway Messages (Part 2)

- Software Engineering promotes a disciplined approach to software development following a ***well defined process development model***
- A number of Software Development Process Models exists with their pros and cons.
- Software Engineering Process defines:
 - *Who* – the roles involved in the software development activity
 - *What* – the activities the roles need to perform to develop the software
 - *When* – the sequencing of activities

Further Reading (Part 2)

- The material in this lecture is ***not covered*** in the recommended textbook. There are numerous sources (including on-line) that can be used.
- I would recommend:
 - Ian Sommerville's "Software Engineering", Edition 6 - 9: Chapter 1, 2.
 - The University library should have a significant number of copies.

Part 3: Software Requirements Engineering

Roadmap

- Why Software Engineering
- Software Engineering
- Software Process Models
- [Software Requirements Engineering](#)
- Object-oriented analysis and design
- Brief introduction to UML

Requirements

A requirement is...

- Something that a product **must do**, must **NOT do** and **handle anomalies** in a particular way (functional requirement), or
- a quality (non-functional requirement) that the product **must have** (Robertson & Robertson 1999). These typically refer to the software as a whole.

Robertson S. & Robertson J., 1999, 'Mastering the Requirements Engineering Process', Addison-Wesley
Jackson M., 1995, 'Software Requirements and Specifications', ACM Press/Addison-Wesley.

Functional Requirements

These are usually split into two broad categories:

- **User requirements**,
 - drawn from the user's viewpoint. These are intended to be useful for **contracts** and more generally as a communication tool between the stakeholders: users (e.g., client organisation/individual) and software developers.
 - Should include **acceptable responses to undesired events**.
- **System requirements**,
 - derived as a result of **analysis**, e.g., modelling the response of the software to external stimuli to clarify the logical structure of the system and **what** it must do (not how)
 - In this module the system requirements will be documented as UML analysis models (use case models, class and sequence diagrams).

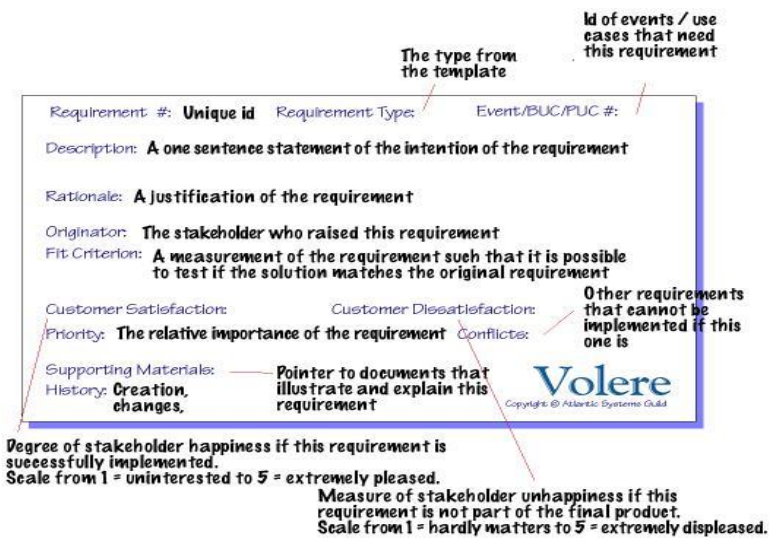
Non-functional Requirements

These can be classified as:

Reliability
Availability
Maintainability
Usability
Recoverability
Safety
Efficiency
Security

"RAMURSES"
(the meaning for these attributes is spelled out in detail in textbooks on Software Engineering and in the SEBOK book)

User Requirements: Volere Template



Example of a User Requirement in Volere Template

Requirement ID: 29	Requirement Type: FR	Event/Use Case #2
Description: The product <i>shall</i> identify all outstanding books on loan to a Borrower		
Rationale: Need to know which book loans are overdue when making a decision about whether to extend a loan.		
Source: Aislinn Weatherspoon, Chief Librarian		
Fit Criteria: The Outstanding Book Loans for a Borrower are those where the Loan Expiry Date is before or equal to Today's Date		
Customer Satisfaction: 5		Customer Dissatisfaction: 5
Priority: Essential		Conflicts: None
Supporting Material: Library loan conventions paper January 2008		<div>Volere</div> <div>Source: Atlantic Systems Guild</div>
History: June 25, 2009 <i>Passed Quality Gateway review</i>		

System/Software Specification

- Specification:
 - in the Oxford English Dictionary (OED): “A detailed description of the particulars of some projected work in building, engineering, or the like, giving the dimensions, materials, quantities, etc., of the work, together with directions to be followed by the builder or constructor...”
- in engineering: a description of what is required from a system/component, given to those who have to deliver it
 - without *unnecessary details* of the system's construction or method of construction : “*what, not how*” (roughly)
 - “delivering” may mean building; buying or selling; modifying an existing system to fix problems or satisfy new needs; etc

The Software Requirements Document/Specification

The software requirements document/specification (SRD/S) is the official **statement** of what is required of the system developers.

Can be anyone (or more) of the following:

- A written document
- A set of models
- A formal mathematical description
- A collection of user scenarios (use-cases)
- A prototype

Should include both:

- a definition of **user requirements** and
- a set of models that capture the **system requirements**.

It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Intended audience (users): system customers, managers, system engineers, system test engineers, system maintenance engineers

IEEE standard 830 Edition 1998

IEEE Recommended Practice for Software Requirements Specifications (SRS)

- | | |
|---|----------------------------------|
| 1. Introduction | 2.2 Product functions |
| 1.1 Purpose of the document | 2.3 User characteristics |
| 1.2 Scope of the product | 2.4 General constraints |
| 1.3 Definitions, acronyms and abbreviations | 2.5 Assumptions and dependencies |
| 1.4 References | 3. Specific requirements |
| 1.5 Overview of the document | 4. Appendices |
| 2. General description | 5. Index |
| 2.1 Product perspective | |

What is a Good Specification?

- Precise
 - tells the “deliverers” what they need to know so that they can deliver the system required
- Correct
 - describes what is really needed
- Clear:
 - easy to grasp and to understand without error for its intended users
 - or “as easy as possible given the complexity of what needs to be communicated”
 - different needs (terseness vs. detail) in different stages of the development chain, e.g.:
 - negotiating **user requirements** between developers and non-technically minded end users, managers
 - giving instruction to programmers with expertise in the **application domain**, or instead lacking this expertise
 - to communicate to a system designer what he/she can expect from a pre-developed (i.e., off-the-shelf) component

What is *Difficult* About Specifying Software-Based Systems?

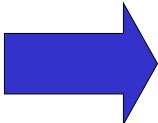
- It is difficult to build a system right
 - Computers do what they are programmed to do
 - If the developer doesn't remember that some special case must be treated specially, the computer lacks the common sense to do so
 - Difficulty of managing *complexity*: the human mind is limited
- Likewise, it is difficult to [decide and then to] communicate clearly what system we want
 - Developers [try to] build what they are *told* to build
 - Difficulty of managing *complexity*
 - Specification is communication, often between different cultures
 - *A typical issue is the common knowledge or legal framework in a particular industry (e.g., automotive industry) but the developers are not aware of it because they are not part of this industry.*

SRD/S and Agile Methods

SRD/S is not used with *agile methods*:

- the requirements change too quickly, and SRD is constantly out of date (maintaining SRD up to date is seen as a waste of time)
 - This is rarely justified, as software typically must be maintained. The lack of documentation makes maintenance difficult
 - There are examples of open-sources projects which, which are poorly maintained due to lack of documentation.
- In extreme programming (XP) the *user stories* are a minimalistic form of SRD
- Agile methods are not universally applicable (e.g., they are *problematic for critical systems*, large software systems likely to be in service for many years, etc.)

Complex systems are hard to understand

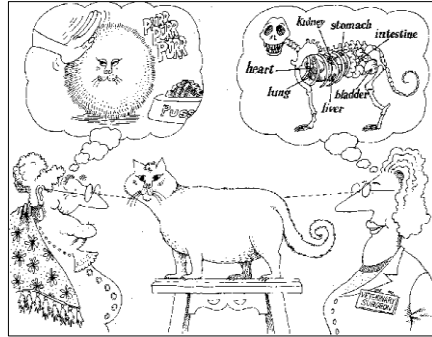
- Hampered by human limitations:
 - cognitive limitations of individuals
 - poor communication between individuals
 - Processes (to deal with complexity):
 - *Abstraction*: removing unnecessary details from a description
 - *Decomposition*: separating an entity into its constituent parts
 - **Abstraction** (i.e., modelling) helps people to understand information and ideas:
 - Grouping
 - Generalising
 - Chunking
- 
- identification of components and subsystems
 - manageable model of the system
- Do it systematically by applying some *process*
 - Use *tools* to *improve productivity* that support modelling and model refinement: *analysis -> design -> implementation*



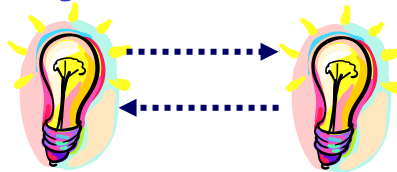
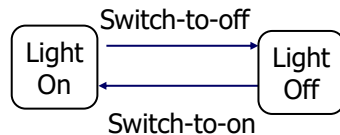
Models and Abstraction

A **model** is a description of a system that:

- **clearly** captures **important** aspects of it from a certain **viewpoint** and for a particular **purpose**, and
- **simplifies** or omits all the other aspects of it systematically
- to **help understanding**, communication, verification



Example: behaviour model for a light switch



Takeaway Messages (Part 3)

- Requirements Engineering is an important part of software engineering
 - Getting requirements wrong leads to faults which are very expensive to fix
- Software Requirements specify WHAT the system should and should NOT do, not how the functionality should be implemented
- Specifying requirements serves the needs of stakeholders
 - *User Requirements* are for clients (typically non-technical personnel) and are stated at a high level of abstraction
 - *System Requirements* are used by system/software developers and are more specific
- Requirements are broadly divided into
 - Functional requirements – related to individual features and may refer to parts of the system
 - Non-functional requirements are specified for the **system as a whole**
- Requirements are captured in a Requirements Specification Document
- Modelling and abstraction are useful ways of dealing with system/software complexity

Further Reading (Part 2)

- The material in this lecture is ***not covered*** in the recommended text book. There are numerous sources (including on-line) that can be used.
- I would recommend:
 - Ian Sommerville’s “Software Engineering”, Edition 6 - 9: Chapter 4.
 - The University library should have a significant number of copies.

Part 4: Object-oriented Analysis and Design with UML

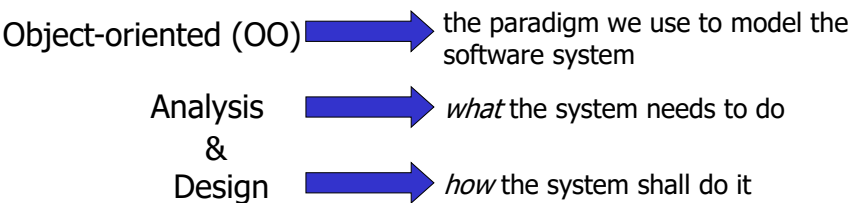
Roadmap

- Why Software Engineering
- Software Engineering
- Software Process Models
- Software Requirements Engineering
- Object-oriented analysis and design
- Brief introduction to UML

What is Object - Orientation?

- A specific *way* of modeling and building software systems
- OO models systems as *sets of objects* that:
 - *Encapsulate* data and function
 - *Interact* with each other by sending messages
- The objects *should* map directly onto things found in the *problem domain*:
 - E.g., in the banking domain, things such as BankAccount, Person, Money etc.
 - Principle of **Convergent Engineering**

What is OOAD?



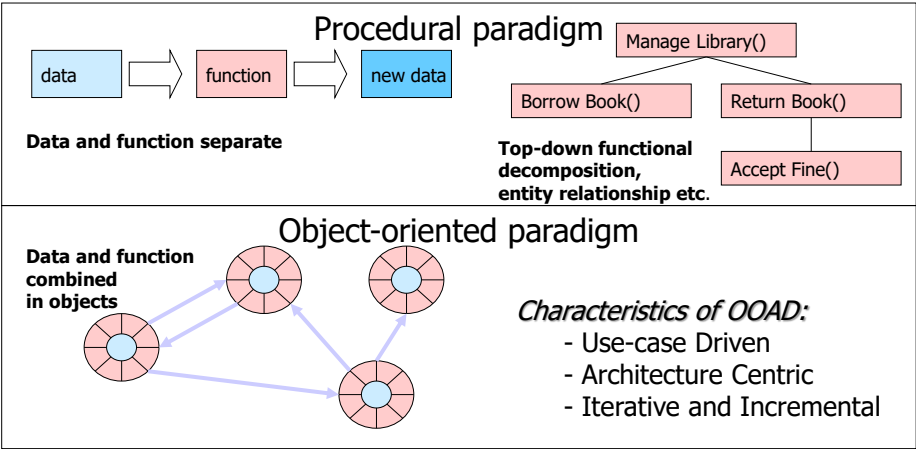
OO – represents the world as interacting objects
OOAD **specifies** software systems in *sufficient* detail so that they can be built
We do this by creating **models** of software systems

- model - a representation of something that captures **important details** from a particular **perspective**

“... all models are wrong, but some are useful”
(George E. P. Box)
Who is George Box: https://en.wikipedia.org/wiki/George_E._P._Box

61

Different paradigms...



- Claims for superiority of Object-oriented paradigm
 - stronger framework
 - reuse of common abstractions
 - resilient under change

62

Roadmap

- Why Software Engineering
- Software Engineering Software Process Models
- Software Requirements Engineering
- Object-oriented analysis and design
- [Brief introduction to UML](#)

OOAD with UML

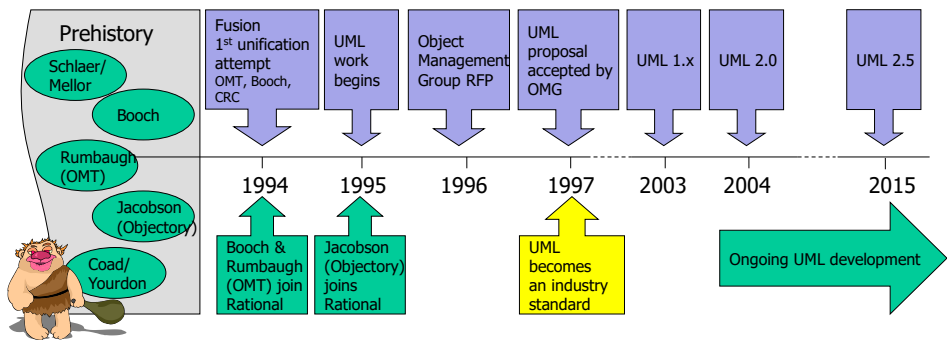
- We will study a set of modelling languages (parts of the Unified Modelling Language, **UML**), suitable for object-oriented software development
 - Focus on analysis and design (OOAD)
 - But linking design models with:
 - software implementation with an OO programming language and,
 - with software testing
- The UML languages have a strong *visual component*
 - Important to learn the syntax and semantics of the UML diagrams to be able to **communicate effectively**
 - Although every complete specification needs *words* (descriptions in natural or formalised languages) as well
 - Writing good specifications requires skills to *write well*.
- Via numerous examples we will learn how to use UML diagrams effectively.

Unified Modelling Language (UML)

- UML is a general-purpose **visual modelling language**
- Can support **all existing process models**
- Intended to be **supported** by CASE (Computer Aided Software Engineering) tools
 - CASE tools address the main objection of the Agile movement - documentation does not have to be a counterproductive **overhead**.
 - With the right CASE tools software development based on UML can actually increase productivity
 - I'll demonstrate some of these benefits during lectures.
- UML unifies past modelling techniques and experience
- Incorporates current best practice in software engineering

UML is not a methodology! It is a visual language!

UML history



The last major upgrade to UML occurred at the end of 2003:

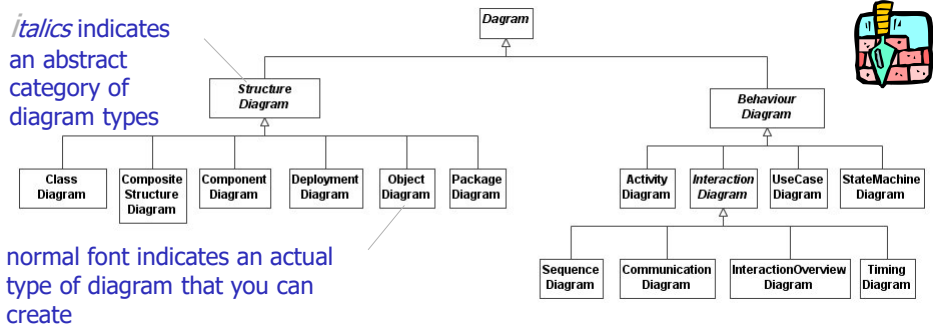
- Greater consistency
- More precisely defined semantics
- New diagram types
- Backwards compatible

Why "unified"?

UML is unified ***across several domains***:

- Across historical methods and notations (documented in books)
- Across application domains
 - banking, process control, energy, telecommunications, avionics, etc.
- Across implementation languages and platforms
 - Many languages (Java, C++, C#, etc.) supported by UML tools, which allows for seamless development.
- Across the development lifecycle and development processes
 - Unified Process (or Rational Unified Process) is typically used with UML, but other, more/less rigid processes can be used too (Waterfall, even Agile)
 - Some of the proponents of Agile manifesto (<http://agilemanifesto.org/>) are well known authors of books on UML (e.g. Martin Fowler)

UML includes 13 types of diagram



Structure diagrams model the structure of the system (the static model)
Behavior diagrams model the dynamic behavior of the system (the dynamic model)
Each diagram gives a ***different view on the modelled system***

UML Structure Diagrams

Represent the data and static relationships in an information system

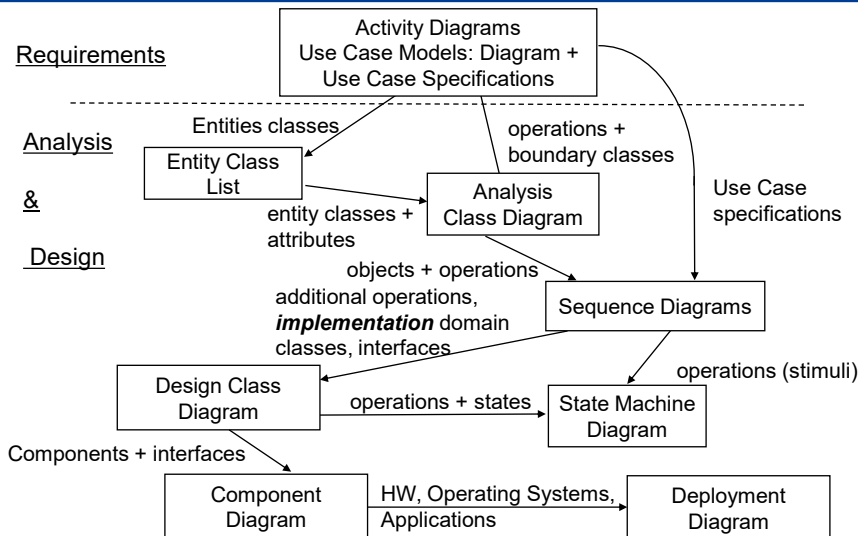
- Class
- Object
- Package
- Deployment
- Component
- Composite structure

UML Behaviour Diagrams

Depict the dynamic relationships among the *instances or objects* that represent the business information system

- Use-case diagrams
- Activity
- Interaction
 - Sequence
 - Communication
 - Interaction overview
 - Timing
- State machines

UML Models and their relationships



IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 71

71

UML benefits

What are the benefits of using UML diagrams at all? A couple of *sceptical* views:

Example 1: Is not software code itself sufficient documentation?

- Not really.
 - Take a large open-source project, e.g. PostgreSQL (Firebird) database server and try to make sense of the 50-100 MB of source code. How can one maintain such large code base?
- Consider also the following:
 - Analysts tend to get paid more than programmers! Why is that? Analysts use models!
 - Many pure programming jobs are outsourced to countries with lower wages. Recently, LLM are used to generate software code
 - A number of our alumni reported to me that knowledge of UML has been a key part in being successful in their early professional career.
- Model – Based Software Engineering (MBSE) is a norm today in *serious IT organisations* (e.g. IBM), leading engineering firms (e.g. Airbus, European Space Agency, etc.)

IN2013 Object-Oriented Analysis and Design

Introduction and context. Slide 72

72

UML benefits (2)

Example 2: "Real programmers do not use models!"

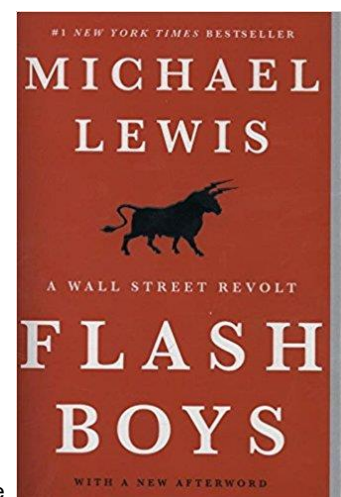
This view is simplistic and often simply **wrong**! The current trend in developing application software shifts towards model driven development! Read about Model Driven Architecture (<http://www.omg.org/mda/>)

Consider also the following aspects:

- With tool support transition from models to coding is **really very simple**.
- Some UML tools support **reverse engineering** of existing source code, e.g. your own code or the code by a 3rd party (Visual Paradigm does it for a number of languages).
 - The integration of Visual Paradigm with popular IDE tools (e.g., Netbeans, Android Studio, etc.) will be demonstrated in the module

UML in industrial practice

- Model-driven development is a reality in the development of **high-integrity** software
 - Airbus makes it mandatory for all its subcontractors to use tools in development (typically UML/SysML based)
 - Papyrus (<https://eclipse.org/papyrus/>)
 - CHESS (<https://www.polarsys.org/chess/>)
 - UML diagrams are used to develop public specifications for interoperable services
- AMI (Advanced Metering Infrastructure), e.g. <http://www.corniceengineering.com/Pubs/AMIUseCaseReportVersion1.0Final.pdf>
- Software development without documentation makes software maintenance difficult.
 - May want to read "Flash Boys: A Wall street revolt".
 - One of the points made in the book is that over the years the code used for **fast computer trading** has become spaghetti code and the lack of documentation made its maintenance very difficult.



Takeaway Messages (Part 4)

- OOAD is a part of Software Engineering
- Modelling is used to deal with systems' complexity
- UML \neq Software Engineering Process (Methodology)
 - UML can be used with any Software Process
- UML is a visual language
- UML diagrams are grouped into:
 - Structure Diagrams
 - Behaviour Diagrams
- UML is a de-facto standard for documenting object-oriented development.
 - Widely used in industry

Further Reading (Part 4)

The material in this part of the lecture is **covered** in the recommended text book, but has been updated to reflect the current status of UML.

Summary

- Software Engineering and OOAD
 - Systems, complexity, requirements, specification
 - Roles of specification: *for communicating ideas*
 - Modelling to deal with system complexity
 - UML \neq Software Engineering Process
-
- Next time:
 - *Use case modelling*