# L1 - Software development: from art to engineering

CS3028 - Principles of Software Engineering

**Ernesto Compatangelo**

Department of Computing Science

UNIVERSITY OF ABERDEEN

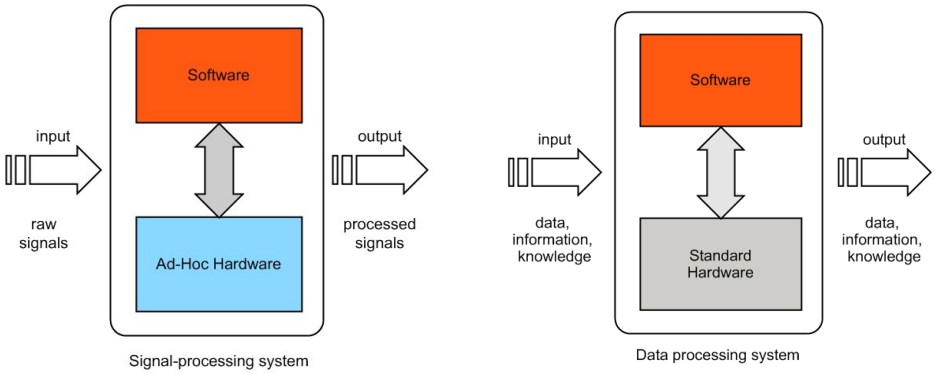


## 1.1 Software systems vs. hardware



### What is a software system?

We can consider two extreme 'patterns':




A number of intermediate cases are also possible

In any case, software plays a paramount role

From the viewpoint of standard engineers, systems are traditionally hardware systems, while the software components are either non existing or not extremely relevant. Conversely, from the viewpoint of software engineers, systems are mostly software systems where the hardware component is controlled managed through standard interfacing protocols.

Until a few years ago, computing mostly focused on data processing systems, with hardware being limited to the usual standard I/O devices (keyboard, printer, monitor, mass memories). However, focus is now changing to heterogeneous, distributed environments where a variety of hardware components interact between them (and with a central software system) through intelligent, autonomous software components that 'represent' the hardware from a computing viewpoint. However, far providing raw signal, such hardware is currently capable of providing structured data through pluggable software drivers that represent the real interfacing protocol for the system developers.

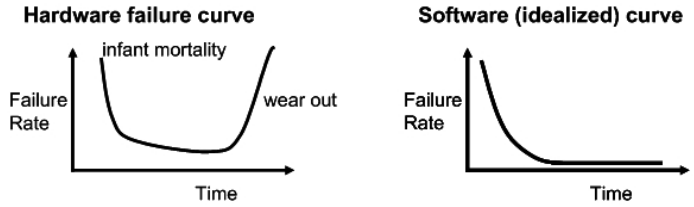
The following three types of software systems are explicitly distinguished due to the different approach used to develop their inner computational structure, their interface and their data storage approach.

- *Real-time (data) control systems*: Patient monitoring systems, flight controllers, most automated factory production systems
- *Information systems*: Database systems, web servers, statistical data analysers
- *Knowledge representation & reasoning systems*: Concept classifiers, rule-based systems, logic-based inferential engines, semantic web systems.

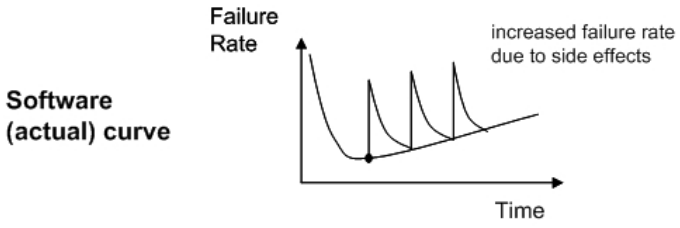


## Unique characteristics of software

- Software does not wear out




- However, software becomes uneconomic to maintain






- Software is developed or engineered, it is not manufactured in the classical sense:
  - Software development and hardware manufacture are fundamentally different;
  - Manufacturing stage introduces quality problems that do not make sense for software.
  - On the other side, software introduces quality problems based on its complexity that do not exist in manufacturing.

- There is now a move towards component-based software assembly (this has been already achieved in hardware since the mass industrialisation of goods production began in the early years of the 19th century)
  - Reusable components allow engineers to concentrate on innovative elements of design;
  - Re-use of software components now emerging;
  - However, most software is still custom built.

## 1.2 Software quality: what is it?

- **Q**: Software quality – same as hardware quality? (hardware quality: when all copies 'are the same')
- **A**: No, each software system is conceptually 'unique' and its physical copies are absolutely identical
- **Q**: How do we define software quality then?
- **A**: We define it in terms of the quality of the development process, which 'is the same' for different software systems
- **Conclusions**: in the software world, it is the quality of the development process that entails the quality of the product. SW development methodologies 'model' the software from its specification to its implementation.

## 1.3 Software: a far-from-success story

Software production has a poor track record — example: Space Shuttle Software

- **Cost**: $10 Billion, millions of dollars more than planned
- **Time**: 3 years late
- **Quality**: First launch of Columbia was cancelled because of a synchronisation problem with the Shuttle's 5 computers
  - Error was traced back to a small change made 2 years earlier and not previously discovered by tests
  - Substantial errors still exist now

## 1.4 Factors affecting software quality

- **Complexity**:
  - The problem domain is complex
  - The development process is very difficult to manage
  - A system is now so complex that no single programmer can fully understand all its code
- **Change**:
  - A system change to fix a bug causes another bug
  - Each change worsens the system structure making the next change even more expensive.
  - As time goes on, implementing a change becomes too expensive and the system is no more viable to maintain

## 1.5 The need for software engineering



### Why Software Engineering (Est. 1968)?

**The 1968 software crisis**: spectacularly wrong estimates leading to unreliable, unmaintainable software ...

A discipline was thus envisaged that explicitly addresses:

- **Economy**: how to produce quality SW *at low cost*
- **Timeliness**: how to produce reliable SW *on time*
- **Quality**: The average software released on the market is far from being error free, so a disciplined approach is needed ...



Software engineering is formally defined as follows.

*The application of engineering to software, i.e. the application of a systematic, disciplined, quantifiable approach to its development, operation, and maintenance*
(IEEE, 1993)

Software Engineering is a collection of concepts, methods and tools that help with the production of:

- a high quality software system
- with a given budget
- before a given deadline
- while change occurs

## What is software engineering?

**A problem-solving approach** based on

- **Concepts**: elements (*e.g.*, packages, classes, attributes) that represent software at a given abstraction level
- **Concept notations**: a textual or graphical set of rules for representing software concepts (*e.g.*, the UML)
- **Activities (*methods in-the small*)**: formal procedures to achieve results using some well-defined notation
- **Paradigms (*methods in-the-large*)**: combinations of activities performed across SW development in a disciplined way
- **Tools**: SW applications to perform one or more activities



Software engineering is based on the notion of software development as a staged modelling activity. During each activity, the product is modelled using different criteria and viewpoints. Five major software development activities have been identified, *i.e.*, Planning, Analysis, Design, Implementation Testing, and Post-Implementation (the latter including deployment, operation, bug-fixing — a.k.a. maintenance).

Development activities based on sound software engineering principles:

- Moves systematically through phases where each phase has a standard set of outputs
- Produces project deliverables
- Uses deliverables (output) from one phase as the input for the subsequent phase
- Results in actual information system
- Uses gradual refinement
- Is based on one software development (i) technique, (ii) methodology, and (iii) set of tools

The lifecycle activity flow and the associated objects are shown in more detail in Figure 1.6.

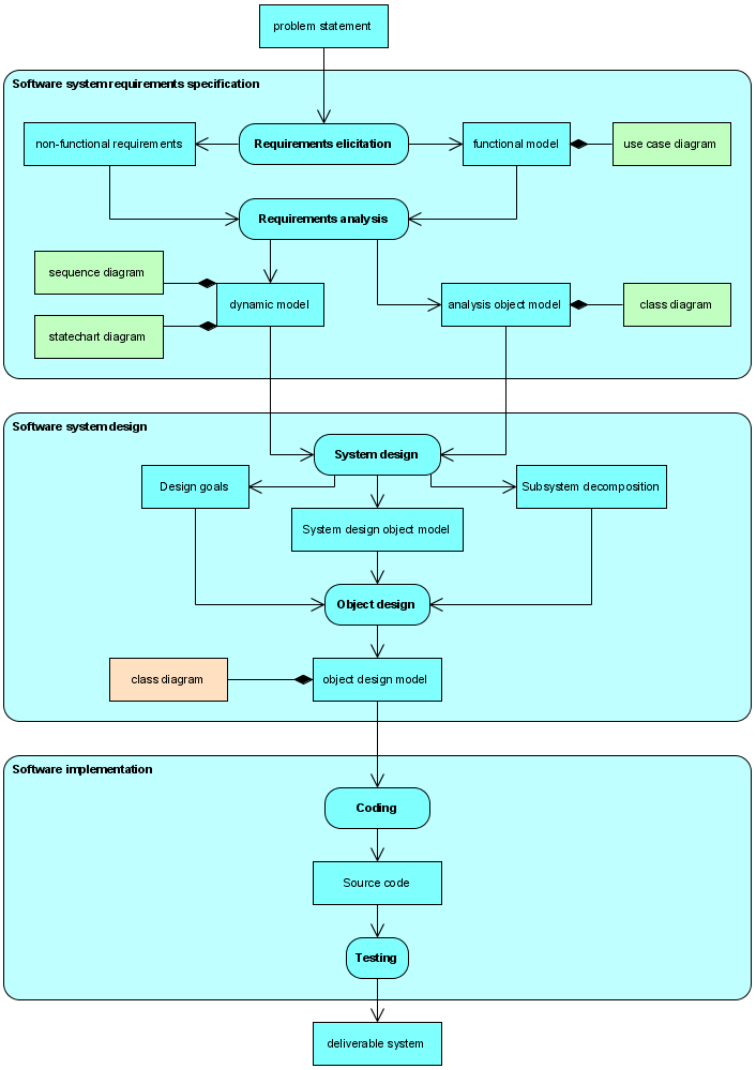



Figure 1: Activity flow in software (i) requirements specification, (ii) design, and (iii) construction

## 1.6 Software development: core issues, activities, and deliverables



### Software development: the core questions

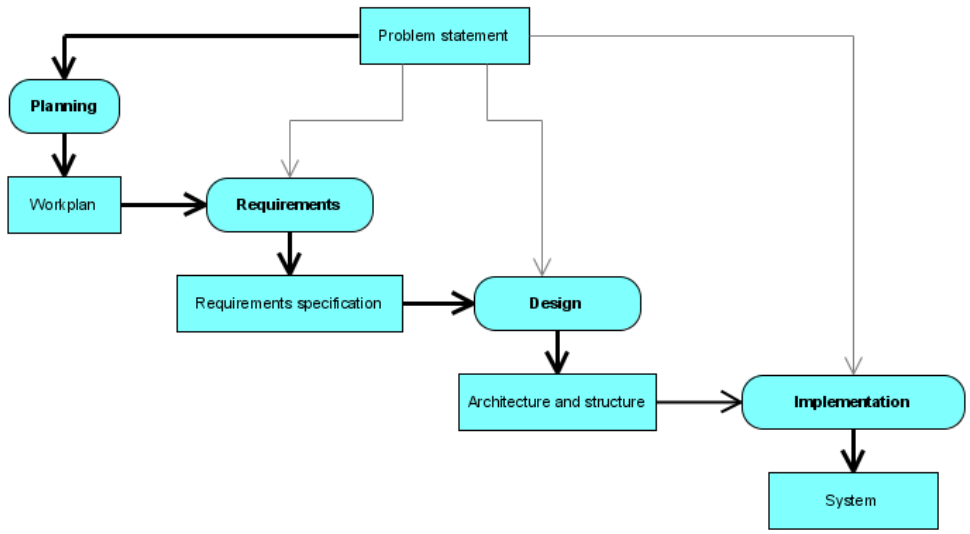
- **Vision & business case**: Why build the SW system?
- **Planning**: How to manage the SW system development?
- **Requirements**: What, when, where will the SW system be?
- **Design**: How will the SW system work?
- **Implementation**: How will the SW code look like?
- **Testing**: How to guarantee that the SW works as expected?

- **Post-Implementation**: How to deploy and 'maintain' the SW?





### Software development: the core activities & deliverables






Software engineering is based on the notion of software development as a staged modelling activity.

During each activity, the product is modelled using different criteria and viewpoints. Five major software development activities have been identified, *i.e.*, Planning, Analysis, Design, Implementation Testing, and Post-Implementation (the latter including deployment, operation, bug-fixing — a.k.a. maintenance).

Development activities based on sound software engineering principles:

- Moves systematically through phases where each phase has a standard set of outputs
- Produces project deliverables
- Uses deliverables (output) from one phase as the input for the subsequent phase
- Results in actual information system
- Uses gradual refinement
- Is based on one software development (i) technique, (ii) methodology, and (iii) set of tools

The lifecycle activity flow and the associated objects are shown in more detail in Figure 1.6.

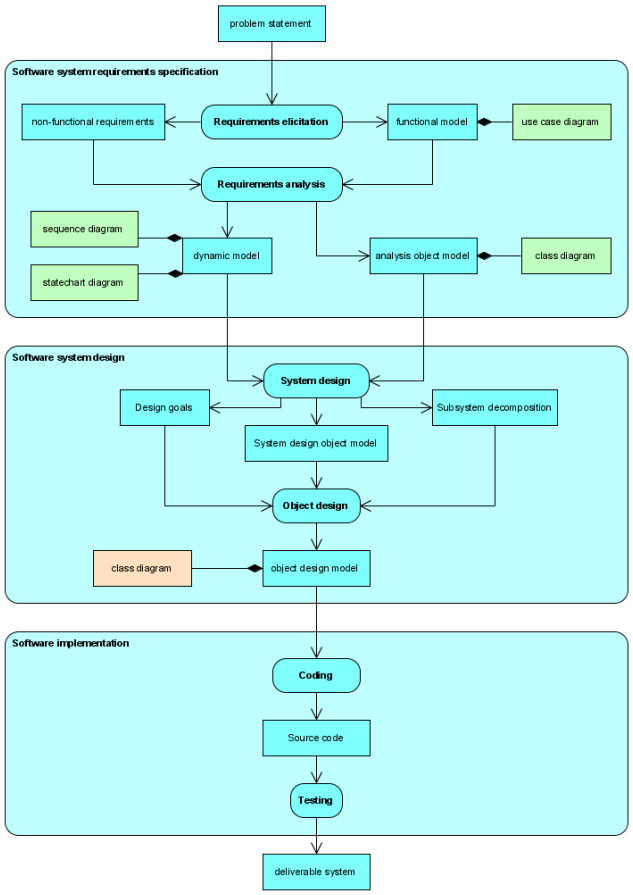



Figure 2: Activity flow in software (i) requirements specification, (ii) design, and (iii) construction

## 1.7 Software development paradigms



### The SW development paradigms

*Paradigms* have been introduced to organise SW development activities.

They all focus on the notion of a SW lifecycle based on some "scenario".

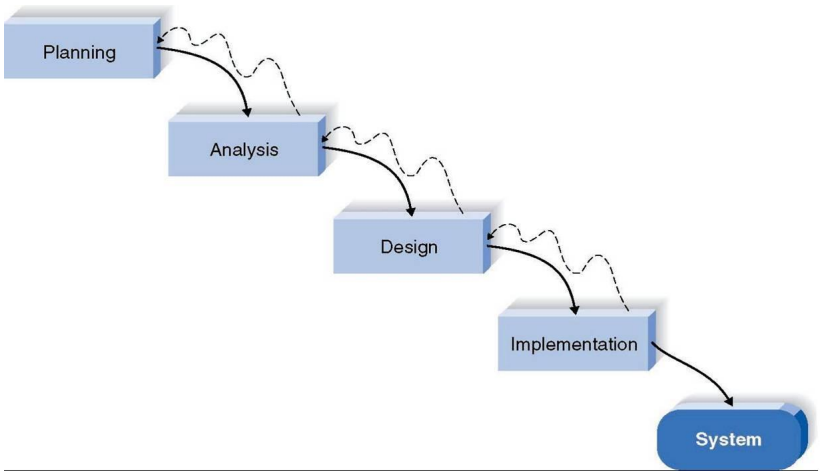
The most important paradigms are:

- Linear sequential (**Waterfall**) paradigm (1970/75)
- **Spiral** (including some prototyping) paradigm (1985/90)
- UML-based **Unified Process** paradigm (1995/2000)
- **Agile** paradigms (2000/2005)





### The waterfall paradigm (analysis = requirements)



## 1.8 Drawbacks of the waterfall paradigm

In a realistic SW development scenario:

- the project feasibility is questionable.
- The customer has only a vague idea of what is required and is unable to give precise input, output and processing specifications.
- The developer often wants to proceed with a "vague idea" on the assumption that "we will fit in the details as we go".
- The customer keeps changing the requirements.
- The developer responds to these changes, making errors in specification and development.

## 1.9 The spiral paradigm in detail

It is based on the following activities for each recurring "phase":

- **Planning**: Define objectives, alternatives, and constraints.
- **Risk analysis**: identify alternatives and risks.
- **Engineering**: Develop 'next-level' product.
- **Customer evaluation**: Assess results of development.

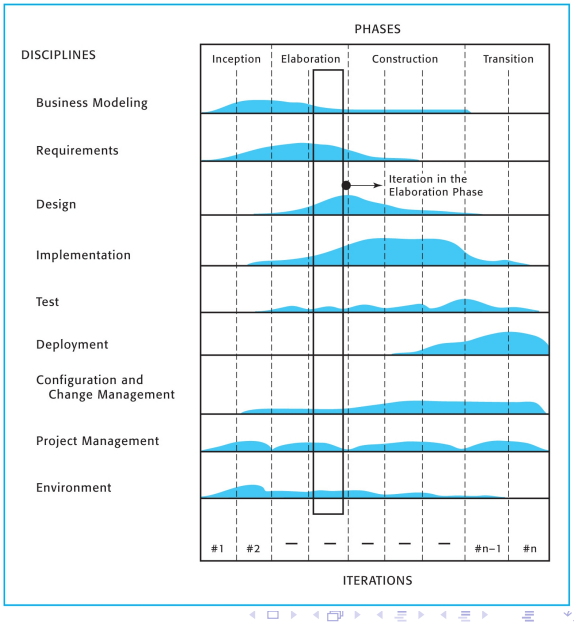
## 1.10 The Unified process in detail



### The Unified Process (UP)

Is organised into:

- 4 *phases*, each making use of
- 9 core *disciplines* (aka activities) and
- *iterations* within each phase.



## Core UP disciplines (activities)

1. **Business modelling**: models the application domain & its workflows
2. **Requirements**: defines the user requirements
3. **Design**: designs the system
4. **Implementation**: writes the software
5. **Test**: tests the system
6. **Deployment**: integrates the software into the organisation
7. **Configuration & Change management**: manages the artifacts of the evolving system
8. **Project management**: manages the development process
9. **Environment**: supports the development with methods and tools





## Unified Process Phases

- **Inception** *(is not just planning and analysis)*: makes the business case; defines some top-level requirements; sketches some critical, central program classes
- **Elaboration** *(is not just design)*: defines *mostly* the system requirements and architecture; sketches the detailed system design; develops some relevant program classes and sketches software testing
- **Construction** *(is not just implementation)*: defines the detailed system design and develops the bulk of the program classes; performs the bulk of software testing; starts deployoment
- **Transition** *(is not just post-implementation)*: deploys the system & integrates it with the using environment; manages software evolution — changes, upgrades

## 1.11 A closer look at some core UP disciplines

### 1.11.1 Business modelling

- Model application domain
- Model domain workflows

### 1.11.2 Project management

- Analyse project feasibility
- Develop work plan
- Staff the project
- Control & direct project

### 1.11.3 Environment

- Decide on activity structure
- Adopt suitable IDEs

### 1.11.4 Requirements

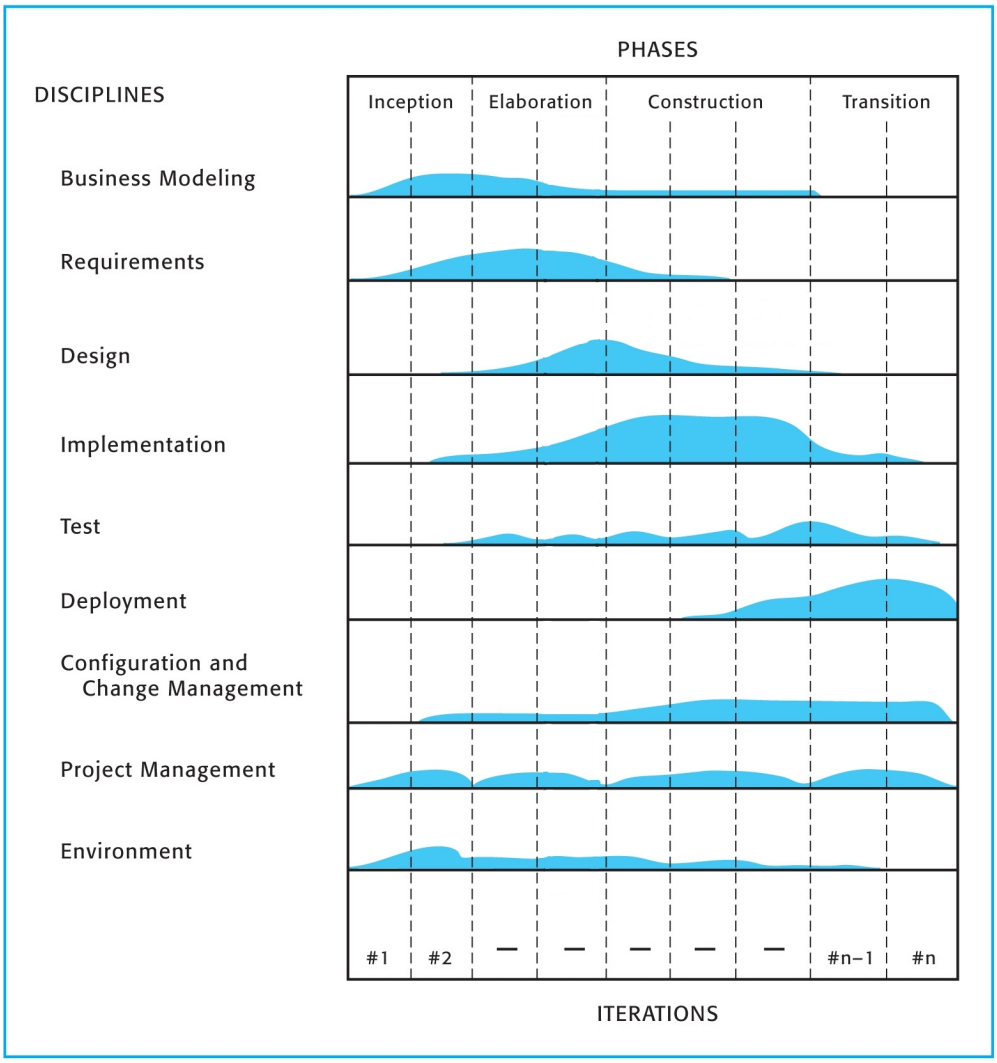
- State vision & business case
- Acquire (elicit) requirements
- Model requirements
- Analyse requirements

### 1.11.5 Design

- Design architecture
- Design HCI
- Design DB and files
- Design program
- Design physical details

## 1.12 How activities (disciplines) distribute across UP phases?

Very roughly, in a typical UP-based project,




- Business modelling =
  I:40%, E:40%, C: 20%, T:0%
- Requirements =
  I:15%, E:75%, C:10%, T:0%
- Design =
  I:2.5% E:30%, C: 65%, T: 2.5%
- Implementation =
  I: 2.5%, E:17.5%, C:70%, T:10
- Testing =
  I:5%, E: 20%, C:45%, T:30%
- Deployment =
  I:0%, E: 0%, C:25%, T:75%
- C&C management =
  I:5%, E: 15%, C:50%, T:30%
- Project management =
  I:20%, E: 20%, C:35%, T:25%
- Environment =
  I:40%, E: 40%, C:15%, T:5%

## 1.13 Preparing for the topics ahead



Next lecture. . .

### UP-based software development: inception

More specifically, we will focus on:

- The *focus* of inception
- The *disciplines* involved in inception
- The *deliverables* produced during inception