Systems and Software Engineering

Lecture 1 – introduction

Jesper Andersson

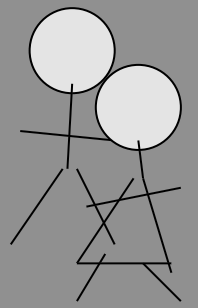
Systems

system
noun sys·tem \ˈsis-təm\
:1 a regularly interacting or interdependent group of items forming a unified whole <a number *system*>
…..

Examples
Software System
System-of- (software) Systems
Cyber-Physical Systems

Someone must take control …
over the system,
over its parts, and
over their interactions and
over the people that develop the system

You should Engineer your systems, not develop.



System and Software Design

- ✓ Characteristics
  - Complex systems
  - Invisible
  - Group activity
  - Always the “first time”
- ✓ Problems
  - Align work
  - Decompose for decentralization
  - Coherence
  - Conformance

System Engineering – Decomposition

- ✓ A system consists of subsystems, which contain subsystems, which can be further decomposed into subsystems …
- ✓ This is true for all systems, biological as well as artificial
- ✓ So problem solving is all about
  1. Finding the best "pieces"
  2. Join them together

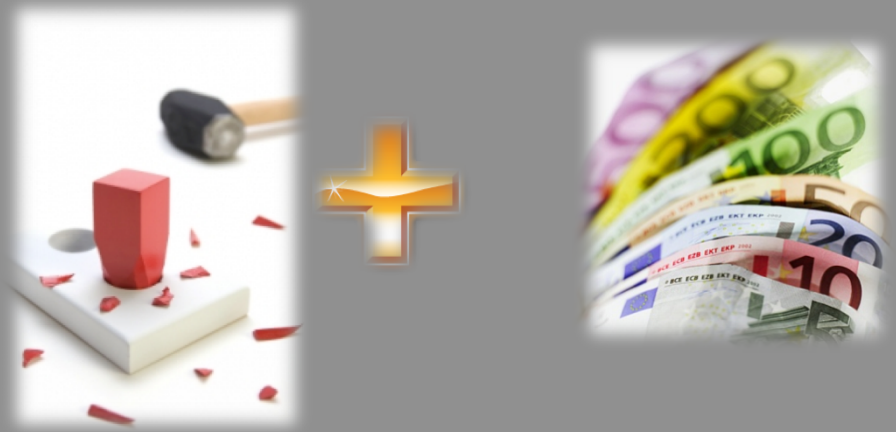
or?

- ✓ Will the pieces fit?

Software Engineering – Challenges

Today we tend to go on for years, with tremendous investments to find that the system, which was not well understood to start with, does not work as anticipated. We build systems like the Wright brothers built airplanes — build the whole thing, push it off the cliff, let it crash, and start over again. (1968, Garmisch, J.W. Graham)

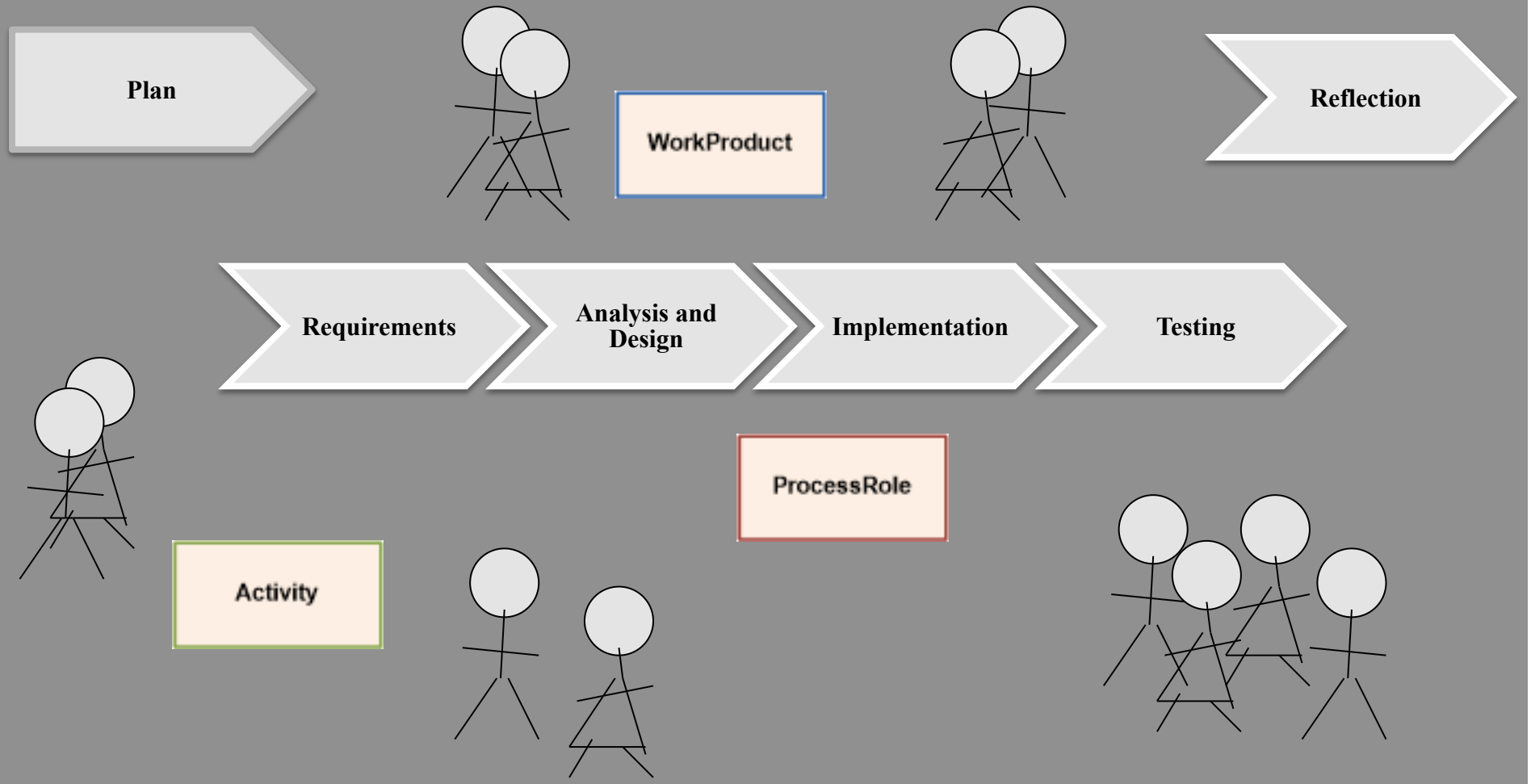
- ✓ Building software is inherently difficult (Brooks mythical man-month)
- ✓ **Essential difficulties**
  - complexity,
  - conformity,
  - changeability, and
  - invisibility



Course Goals

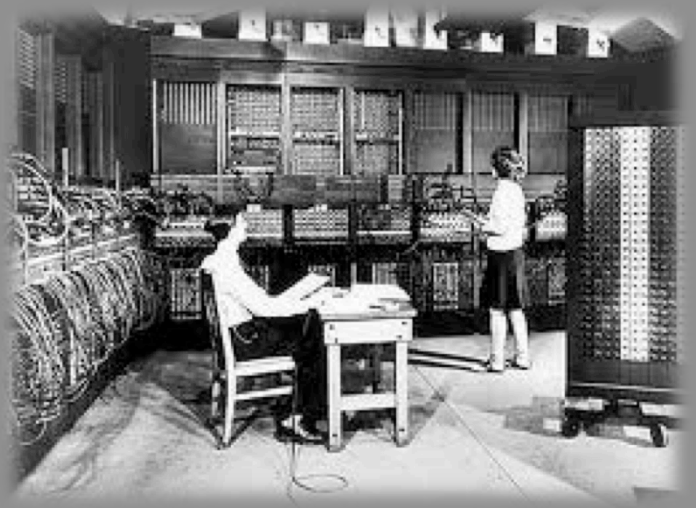
- ✓ How to develop **high-quality** software systems.
  - How to create, refine, maintain adequate models
  - How to implement and verify them
  - How to organize development projects

Course goal: Understand a Process



Software – from Craft to Engineering

- ✓ 1940-talet problem solving with programming
- ✓ Increased complexity
  - More complex problems
  - More complex solutions
- ✓ 1965 Software crisis
  - Over budget
  - Too late
  - Low quality, not inline with expectations
  - Difficult to manage
- ✓ From craft to engineering! **Or**???



Software Development as Craft

craft
(krft)*n.*
1. Skill in doing or making something, as in the arts; proficiency.
2. Skill in evasion or deception; guile.
3. a. An occupation or trade requiring manual dexterity or skilled artistry.
b. The membership of such an occupation or trade; guild.
4. *pl.* **craft** A boat, ship, or aircraft.

- ✓ System and Software Engineering is predominantly an intellectual activity
- ✓ Result dependent on individual's skill
- ✓ Programmer – Craftsman
- ✓ Apprenticeships

or Engineering

Engineering
(ĕn'jə-nîr'ĭng)
n.
1. The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.
2. The profession of or the work performed by an engineer.

- ✓ Predictable
- ✓ Repeatable
- ✓ Transferable

But what are we doing? Well, we solve problems

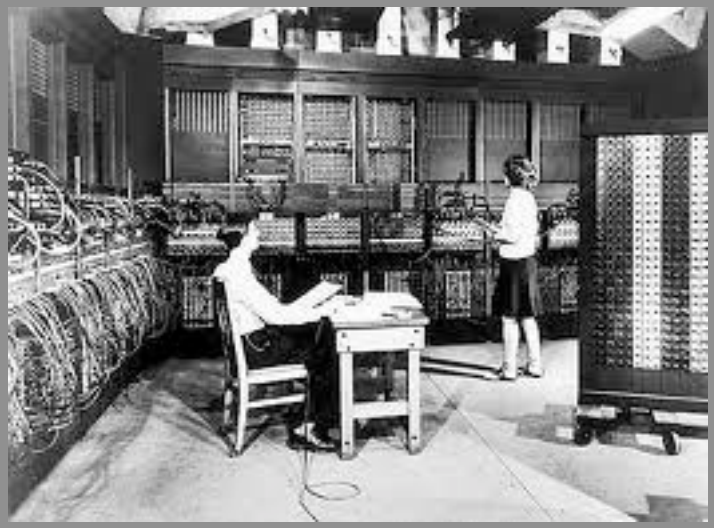
- ✓ Polya's problem solving process

  1. Understand the problem
  2. Devise a plan for solving the problem
  3. Carry out the plan
  4. Evaluate the result

- ✓ Problem solving with *software*
  - The solution is software!
  - We use software specific methods

Software – from Craft to Engineering

- ✓ 1940-talet problem solving with programming
- ✓ Increased complexity
  - More complex problems
  - More complex solutions
- ✓ 1965 Software crisis
  - Over budget
  - Too late
  - Low quality, not inline with expectations
  - Difficult to manage
- ✓ From craft to engineering!



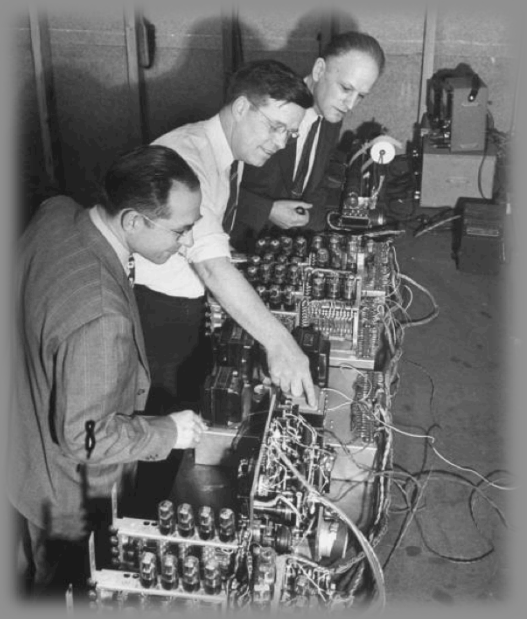
Engineering

Engineering
(ĕn'jə-nîr'ĭng)
n.
1. The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.
2. The profession of or the work performed by an engineer.

- ✓ Predictable
- ✓ Repeatable
- ✓ Transferable

Engineering principles

- ✓ Engineers **apply** the sciences of physics and mathematics to find suitable solutions to problems or to make improvements to the status quo.
- ✓ If **alternatives exist**, engineers **compare design alternatives** and **choose** the solution that **best matches** the requirements.
- ✓ Engineering should be **predictive**, i.e., engineers attempt to predict how well their designs will perform to their specifications prior to full-scale production.



Software Technology – Software Engineering

Software engineering (**SE**) is the profession concerned with specifying, designing, developing and maintaining software applications by applying technologies and practices from computer science, project management, and other fields.

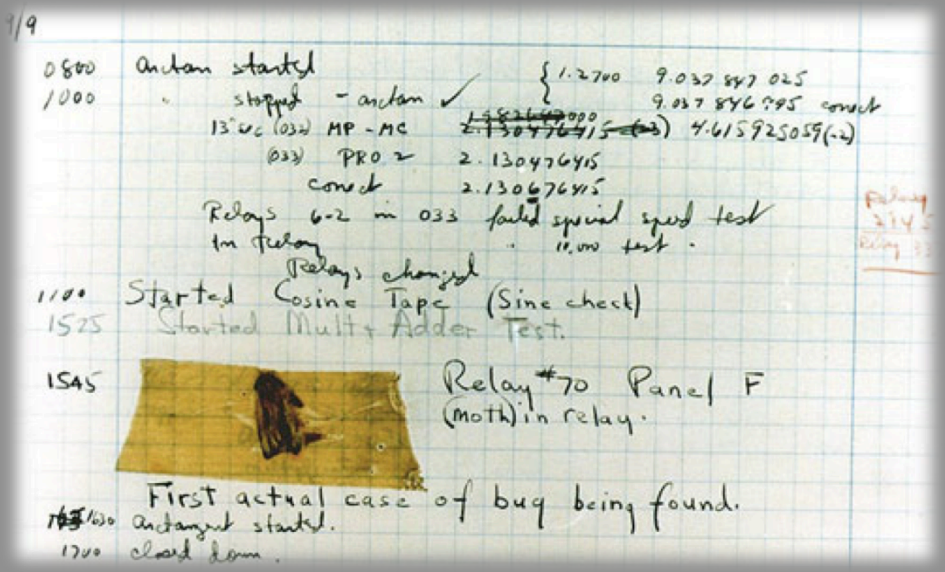
IEEE *Software Engineering Body of Knowledge (*SWEBOK)

- ✓ Industrial-strength software development
  - Methods
  - Techniques and
  - Tools

- ✓ Programming in the small vs. Programming in the large.

System and Software Development Projects
They fail from time to time

- ✓ Denver Airport – Baggage handling
- ✓ London – Ambulance dispatcher



Root cause?

Communication!

A typical software development project

- How the customer described it.

A typical software development project

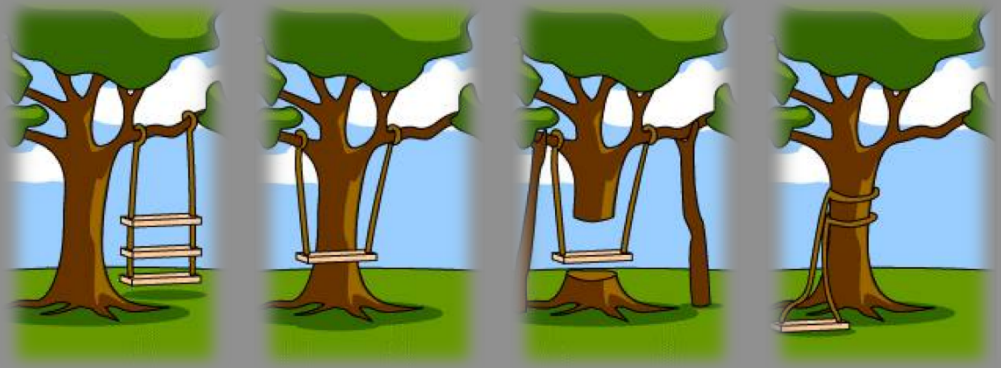
- How the manager understood it.

A typical software development project

- How the architect designed it.

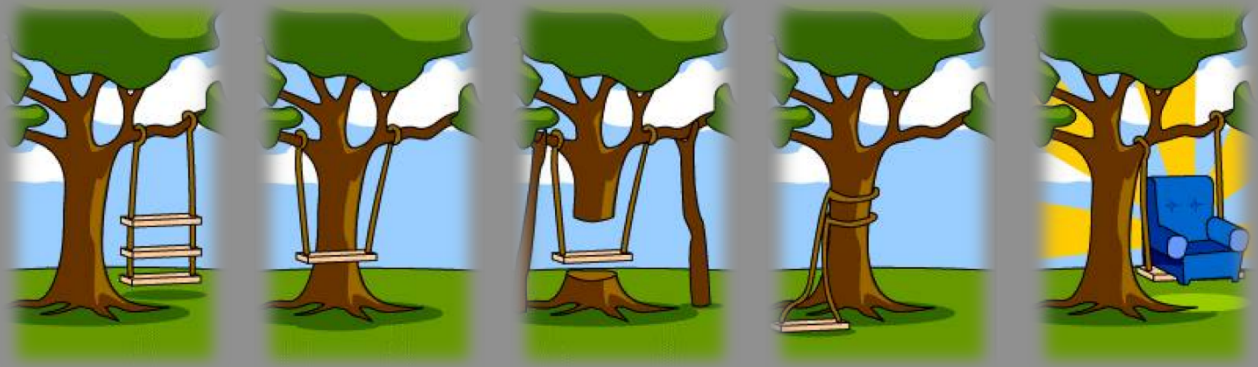
A typical software development project

- How the developers implemented it.



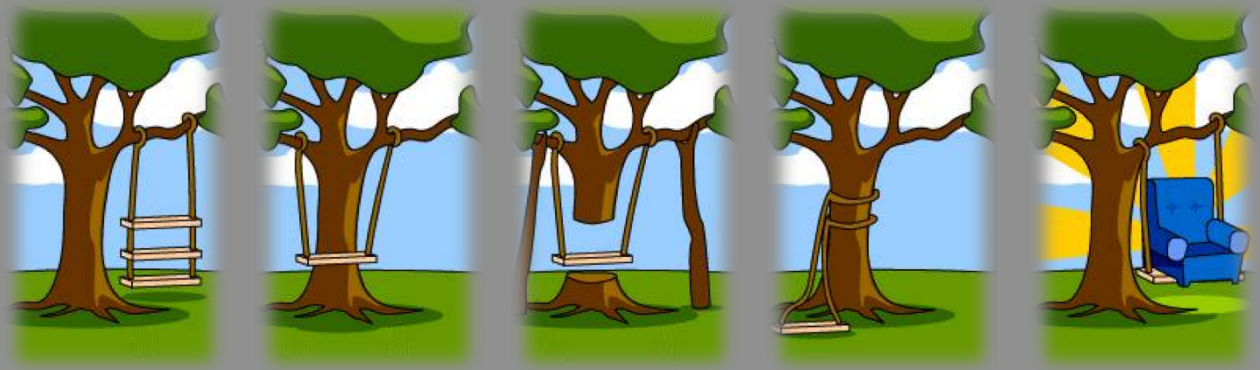
A typical software development project

- How it was marketed.



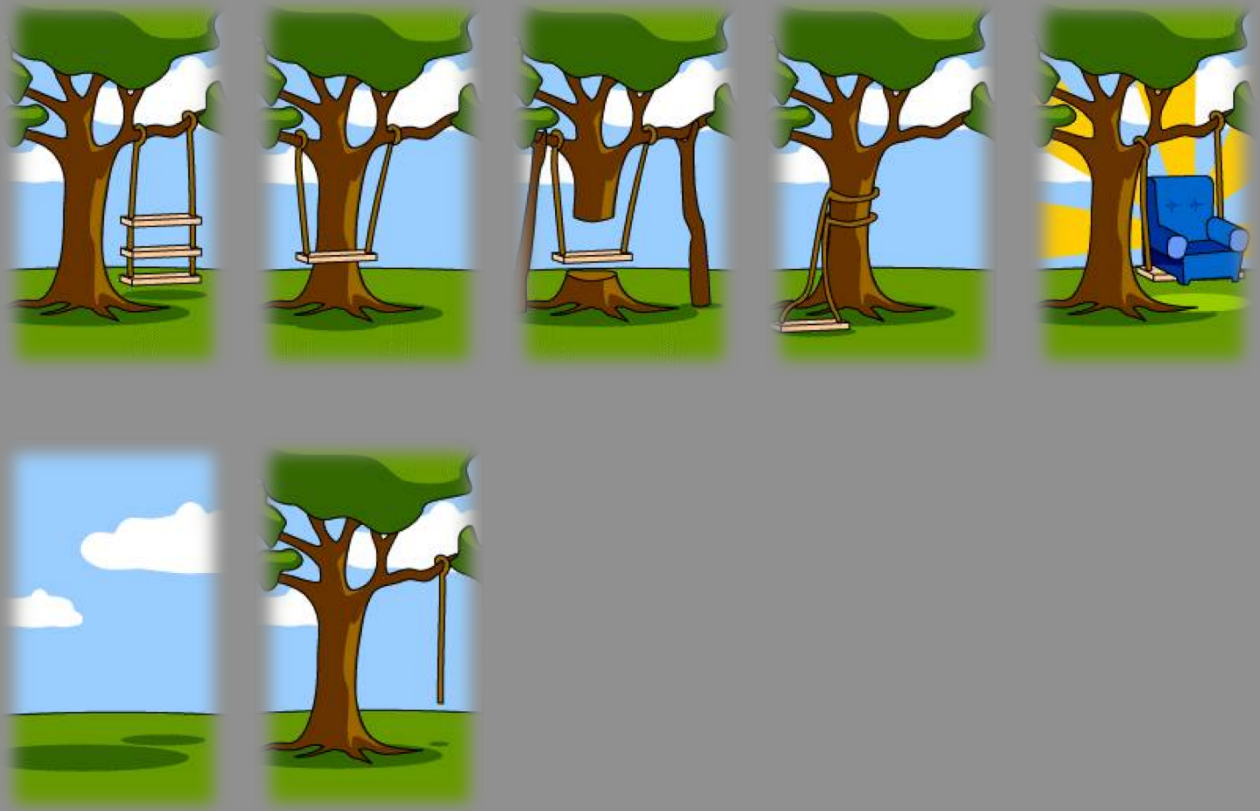
A typical software development project

- How it was documented



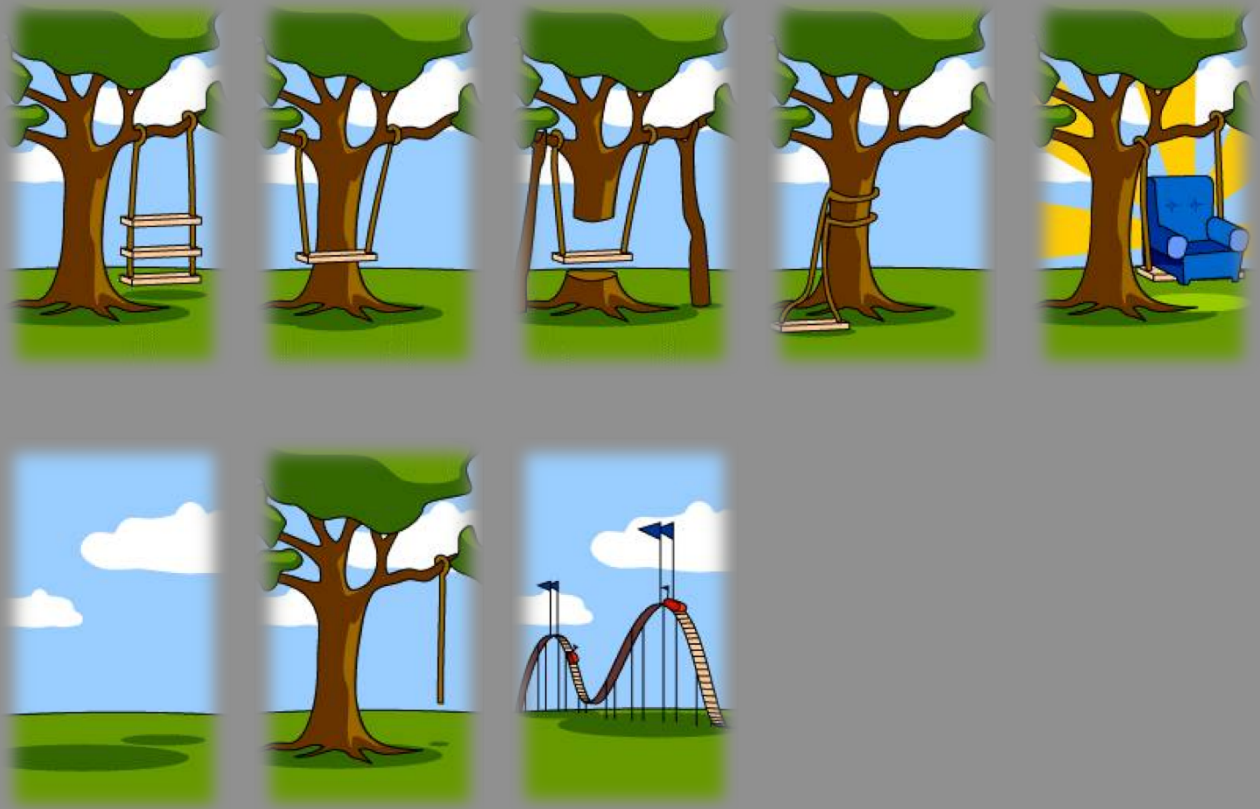
A typical software development project

- What was delivered.



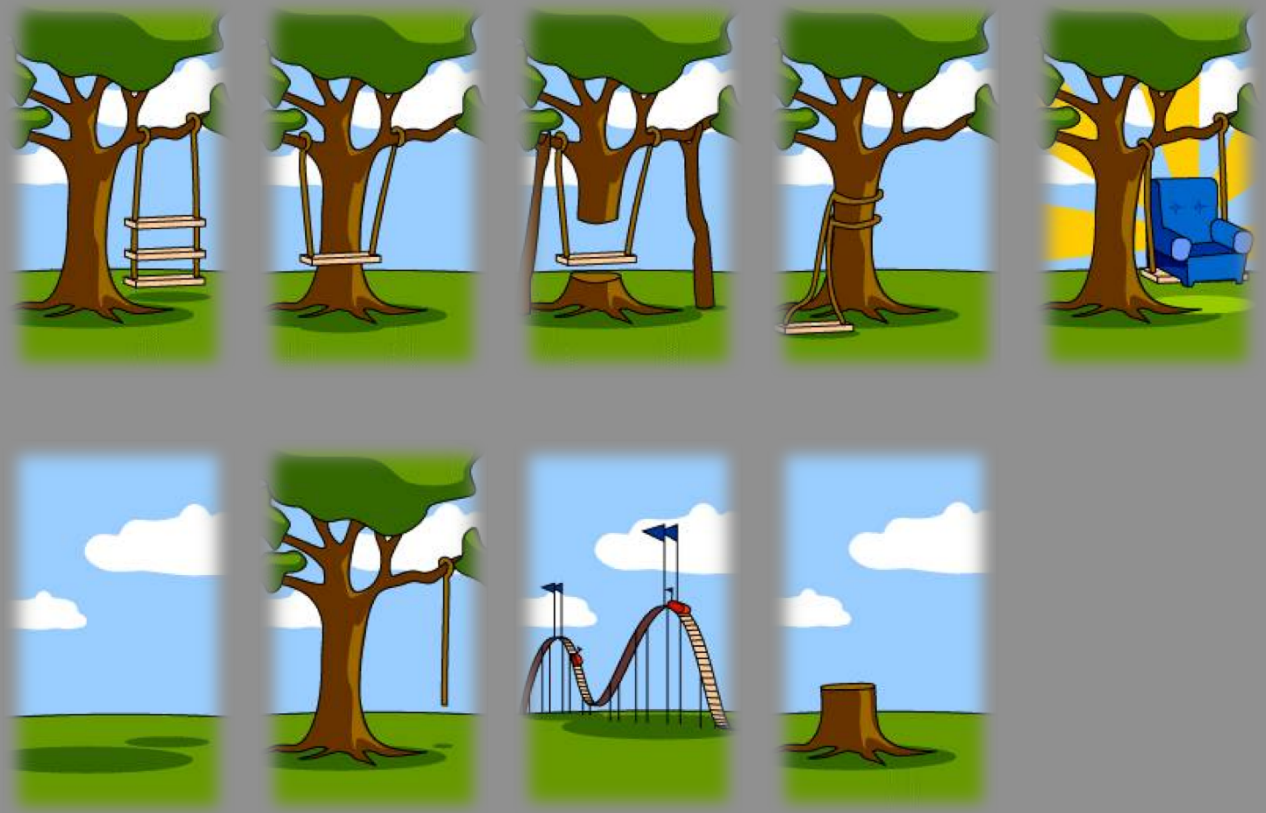
A typical software development project

- What the customer pays for.



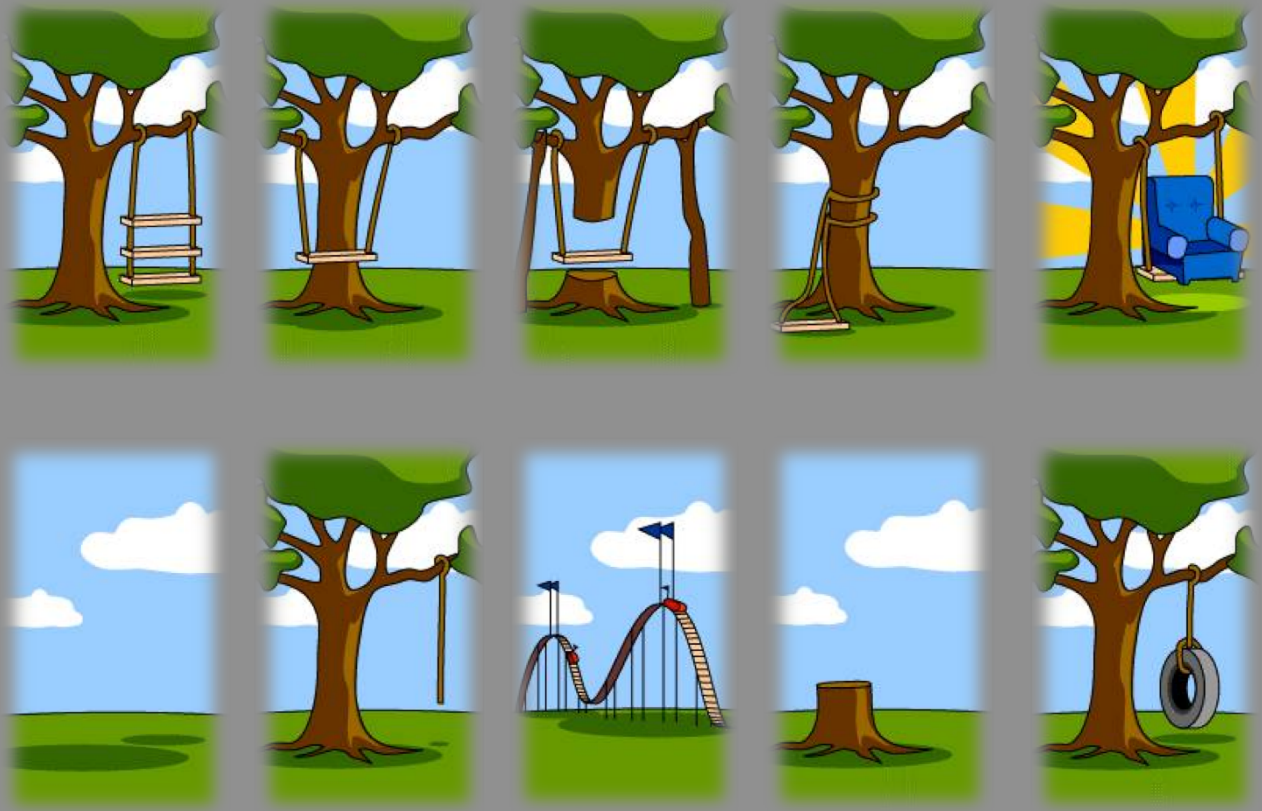
A typical software development project

- How the system was supported post-deployment?



A typical software development project

- What the customer really needed.

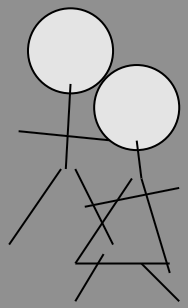


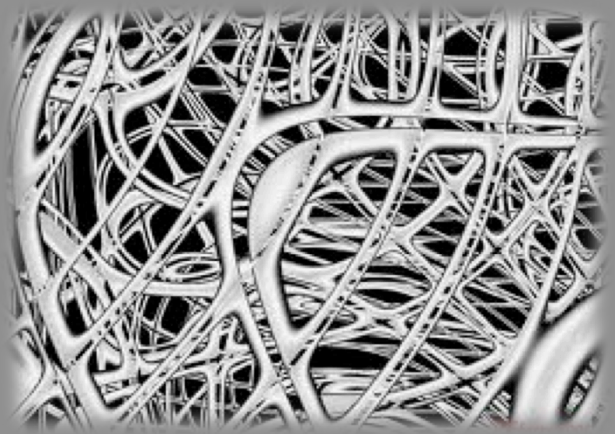
Difficulties in Essence of Software

- ✓ *Inherent complexity* – Due to component interaction, number of possible states grows much faster than lines of code.
- ✓ *Changeability* – Software models the real world. And it changes! Thus, software must change, e.g., more functionality, run on new hardware.
- ✓ *Conformity* – A system must interface with existing (surrounding) systems.
- ✓ *Invisibility* – Cannot visualize software completely or all aspects at once.

How do we manage complexity??

- ✓ Abstraction
  - Simplification
  - Description of entities with required properties (context dependent).
  - Some examples, Data abstraction, Instructions abstraction
- ✓ Decomposition
  - The problem is decomposed in sub problems, which are further decomposed into sub problems …
  - The problem divides the solution into parts.
  - Divide n' conquer



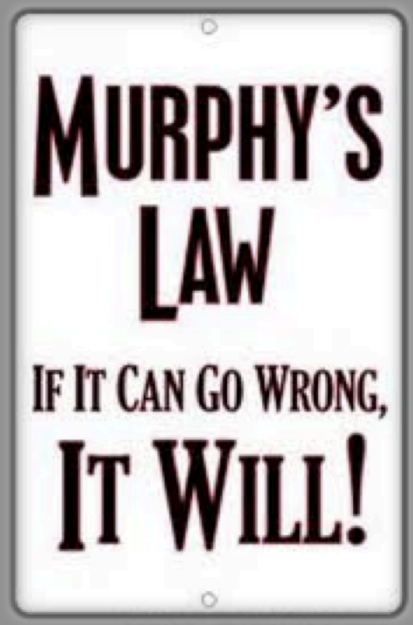


How do we manage changeability?

"If you are not moving at the speed of the marketplace you're already dead – you just haven't stopped breathing yet"

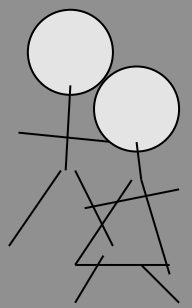
Jack Welch, CEO, GE , 1981-2001

- ✓ Software processes
  - Structure
  - Roles and responsibilities
- ✓ Software models
  - Comprehension
  - Understanding
- ✓ Risk management … if some thing happens … we have at least planned for it.



How do we manage invisibility?

- ✓ Models
  - For requirements
  - For design
  - For implementations
  - For install and operations
- ✓ Models support communication
- ✓ We establish a shared *language* with a *vocabulary and semantics*

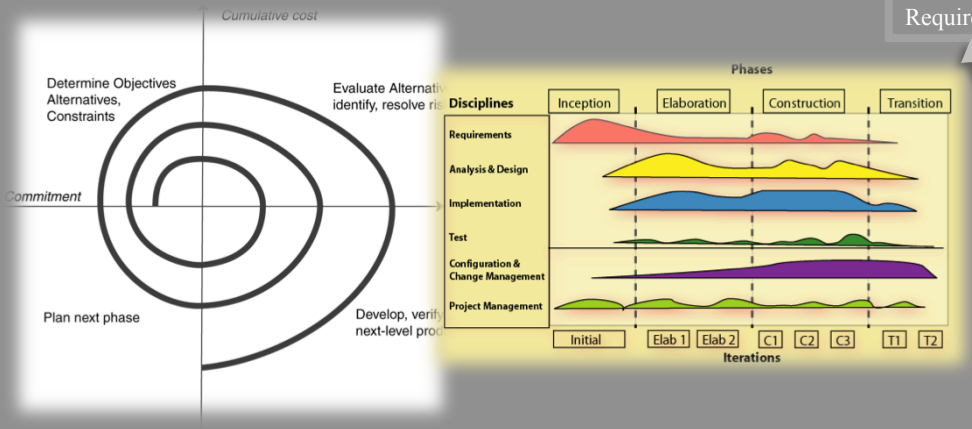


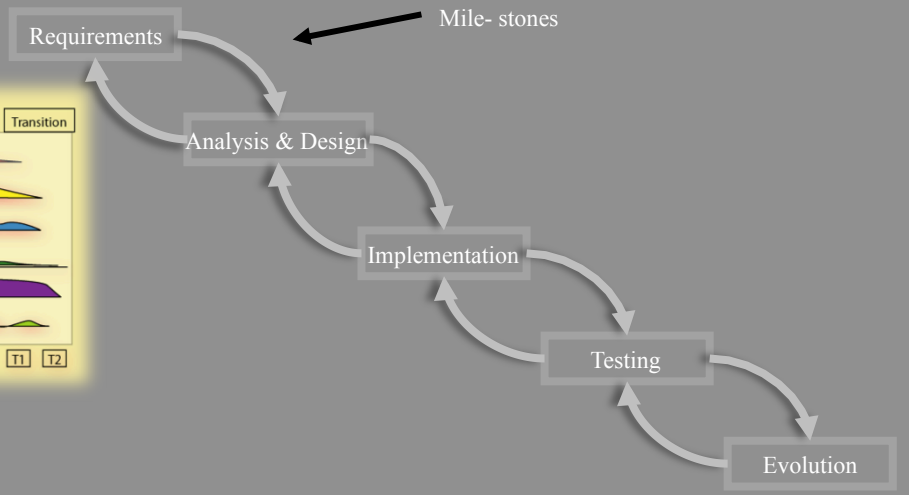
Process activities

- ✓ Requirements Engineering
- ✓ Software Design
- ✓ Software Testing
- ✓ Configuration Management
- ✓ Implementation
- ✓ Project management
- ✓ Software Quality Management
- ✓ Software Product Management

Organize Activities in a Process

- ✓ Waterfall – Royce (1970-ies)
- ✓ Spiral – Boehm (1980-ies)
- ✓ Unified Process (1990-ies)
- ✓ Agile Processes (2000-ies)





Waterfall model (1970)

MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

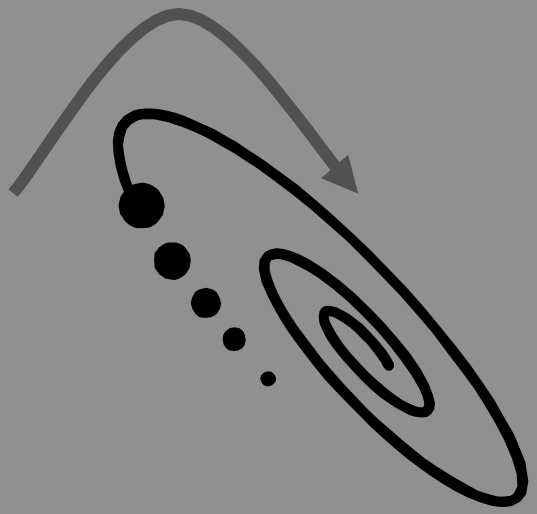
I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on-time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

COMPUTER PROGRAM DEVELOPMENT FUNCTIONS

There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step as depicted in Figure 1. This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it — as is typically done with computer programs for internal use. It is also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product. An

Process models – “forces”

- ✓ Mile-stones
- ✓ Iterations
- ✓ Increments
- ✓ Risks



- ✓ A force your guide to divide the work up in chewable pieces

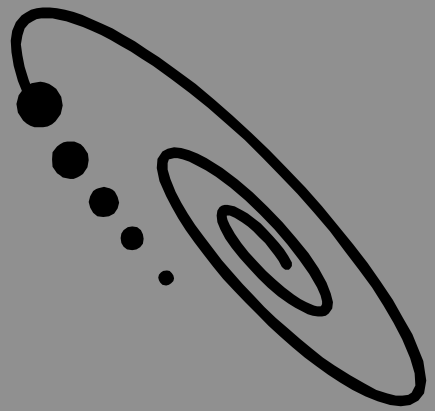
Mile-stones

- ✓ Document driven, this artifact is ready and signed off...
- ✓ Release driven, “this release is done” …

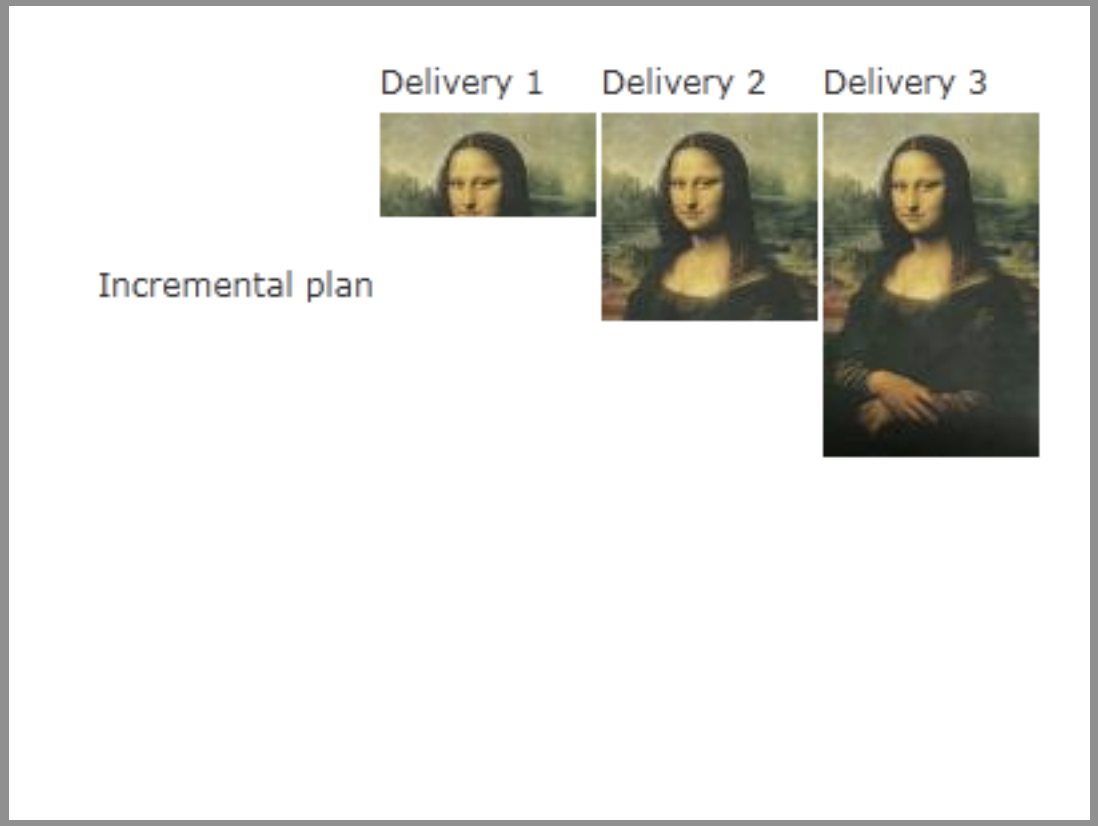
- ✓ **Perspective:** *Who is doing what and when is it done!!*

Iterations & Increments

- ✓ Easier to plan (!?)
  - Smaller steps
  - Shorter time-frames
  - Easier to verify and validate
  - Faster recovery from faults and bad decisions.



Iterative vs. Incremental approaches



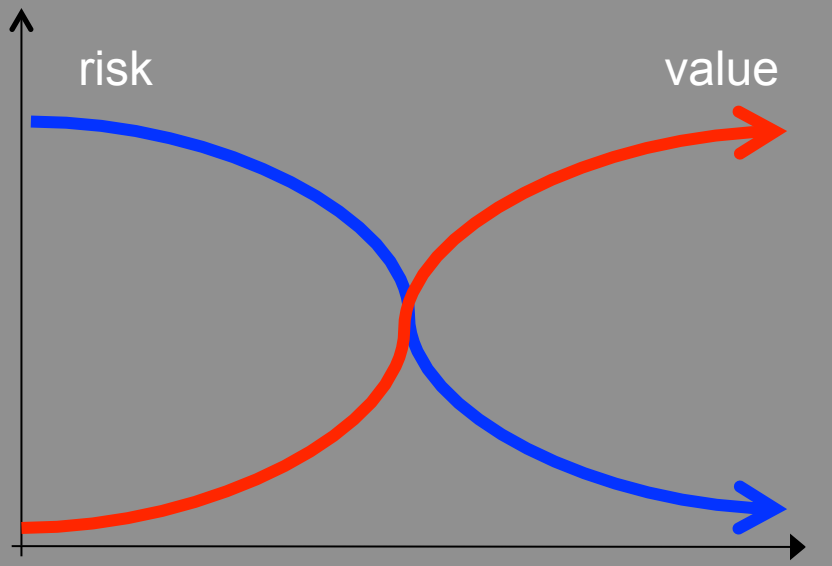
Source: http://www.applitude.se/

Risk management

- ✓ **Proactive** activities to **minimize uncertainties** and **potential effects** on a development project.

- ✓ Decisions based on facts not on assumptions and guessing
- ✓ Continuous activity throughout the system's life-cycle

- ✓ Some risk types
  - product size,
  - market,
  - customer,
  - process,
  - technologies,
  - staff(number of & experience),
  - planning
  - costs



Today's takeaways

- ✓ System and Software

Someone must take control …
over the system,
over its parts, and
over their interactions and
over the people that develop the system

- ✓ Engineering principles and Challenges for software development projects
- ✓ The process – how activities are organized
- ✓ Forces, why a certain organizational principles is applied

Next lecture

- ✓ More on software development processes
- ✓ Process activities
- ✓ Process roles

and

- ✓ Planning