# Rational Unified Process & Designing Software (LL Chapter 9)

RUP pictures in this presentation © IBM/Rational

## Agenda

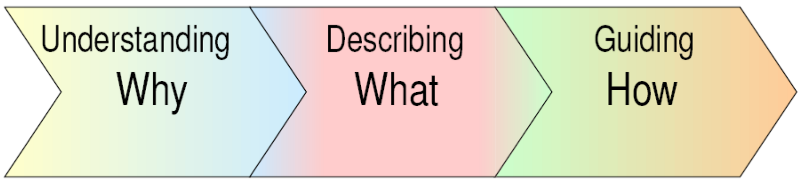
- Recap Architecture
- RUP
- Design heuristics & guidelines

**This afternoon werkcollege**

- Design

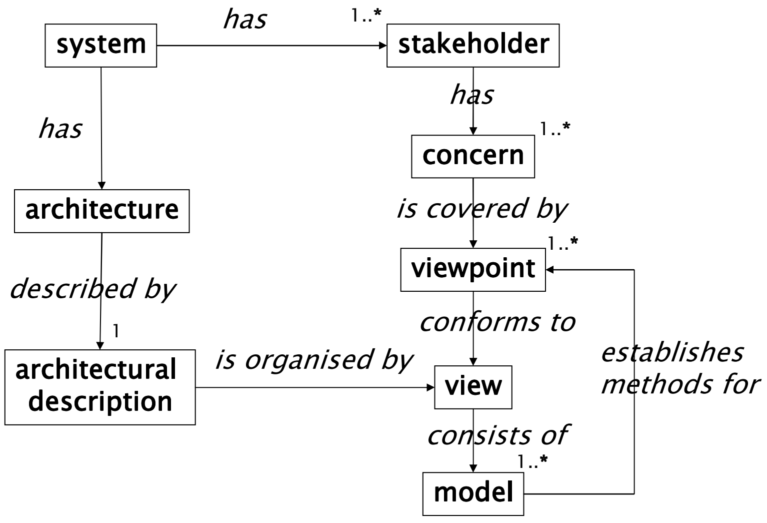
## Multiple Purposes of Architecture

**Understanding**
**+ Analyzing + Communicating + Constructing**

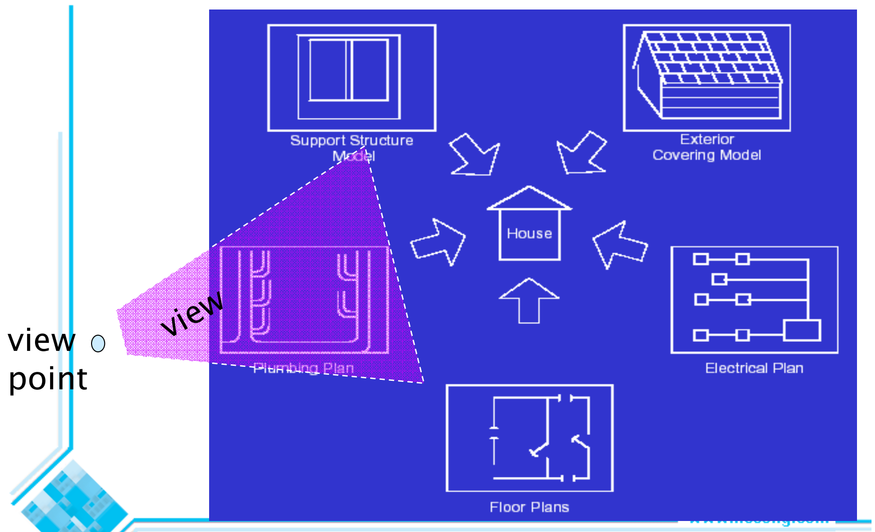


Picture from Gerrit Muller, How to Create a Managable Platform Architecture

## Overview (According to IEEE 1471)



## Viewpoints & views

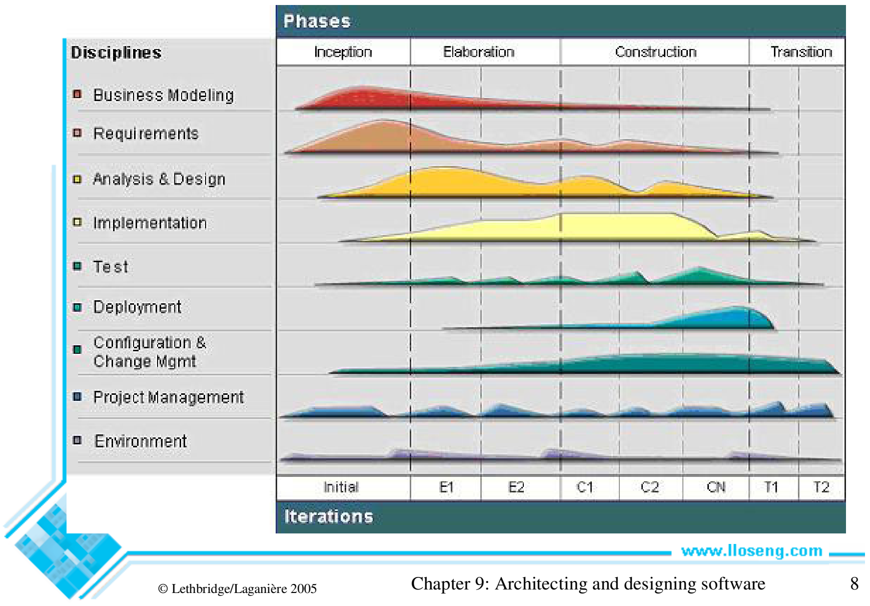


## Recommendations for Architecture Description

- describe the system **goals** & the **assumptions on the environment**
- describe the design **principles, decisions, guidelines**
  - and their **rationale**
- describe **several views** that can be combined in a consistent model at least the following views should be given:
  - **functional/structural (decomposition) view**
    - include a description of the interfaces between (sub)systems
  - **process/dynamical/behaviour view**
  - **deployment view**
- prevent mixing of views
- address **non-functional** (*ilities) aspects
- use a well-defined notation and include its **key/legend**
    - this aids systematic use of notation/avoids inconsistent use
    - improves common understanding
    - prevents mixing of different levels of abstraction
- add explanation in **natural language**

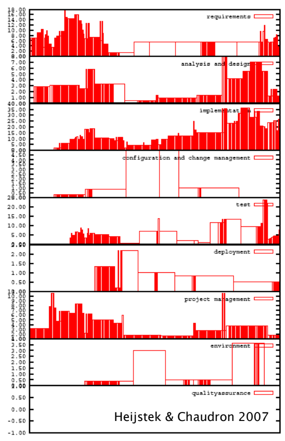
## Rational Unified Process (RUP)

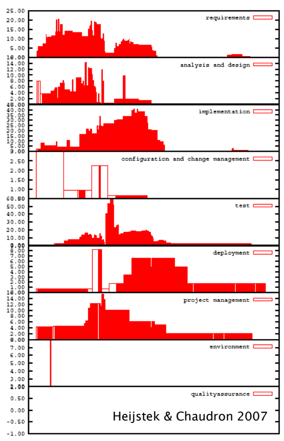
## Rational Unified Process

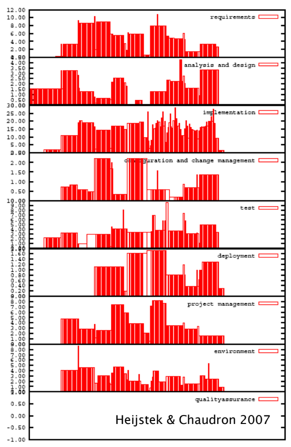


Effort Distribution in Model-based Development – Heijstek & Chaudron

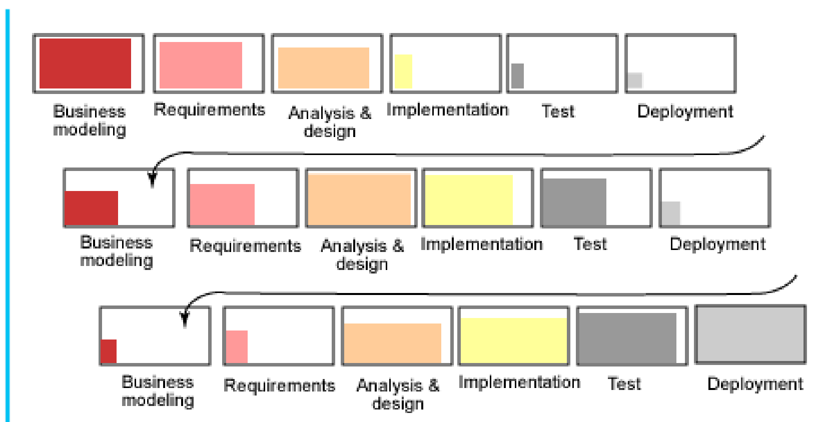
## RUP Humps from 3 (largish) projects



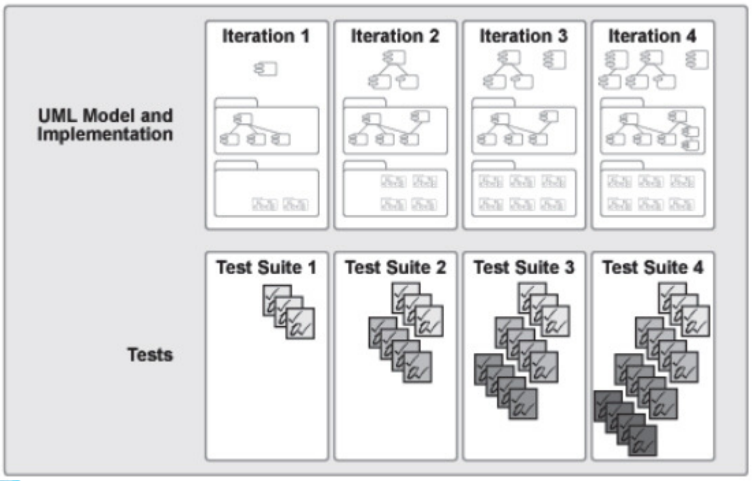




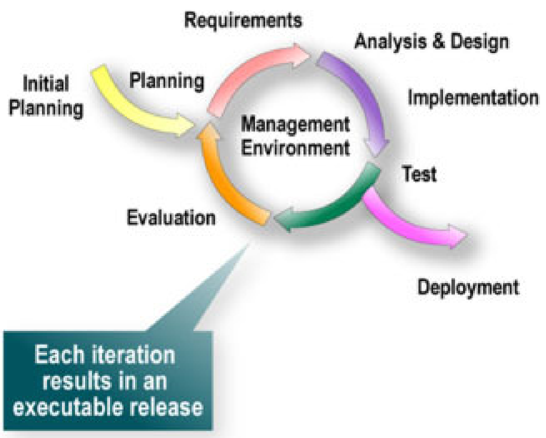
## Progress perspective



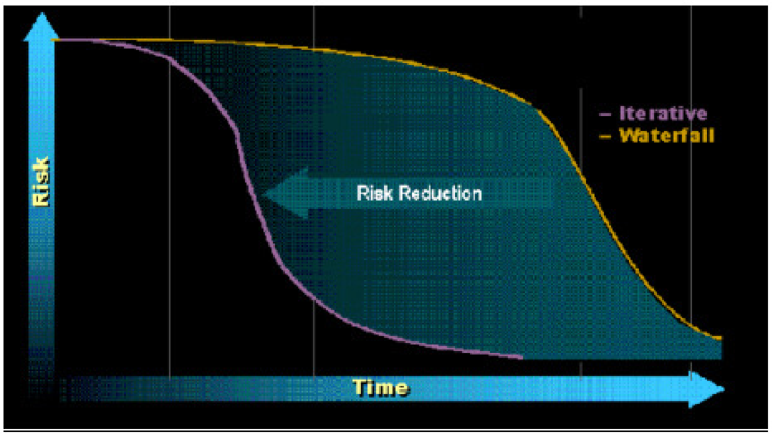
## Progress perspective (alternative pic)



## Iteration Perspective



## Incremental ➔ Risk reduction



## Essentials of RUP

*1. Develop software iteratively; Incrementally build and test*
*2. Manage requirements*
*3. Use component-based architectures*
*4. Visually model software*
*5. Verify software quality*
*6. Control changes to software*

**•Develop a Vision**
**•Manage to the Plan**
**•Identify and Mitigate Risks Early and regularly**
**•Examine the Business Case**
**•Provide User Support**

## How Much Process is Necessary?

Simple upgrades
R&D Prototypes
Static web apps

Dynamic web apps
Packaged applications
Component based (J2, .Net)

Legacy upgrades
Systems of systems
Real-time, embedded
Certifiable quality

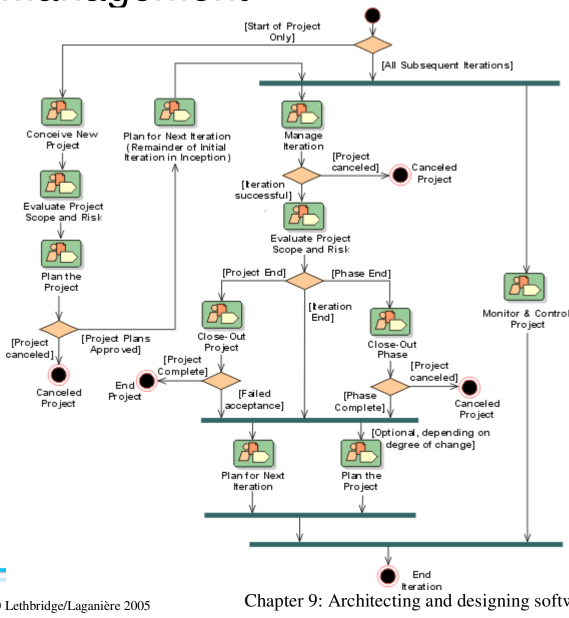
**Strength of Process**

**When is Less Appropriate?**
- Co-located teams
- Smaller, simpler projects
- Few stakeholders
- *Early life-cycle phases*
- Internally imposed constraints

**When is More Appropriate?**
- Distributed teams
- Large projects (teams of teams)
- Many stakeholders
- *Later life-cycle phases*
- Externally imposed constraints
  - Standards
  - Contractual requirements
  - Legal requirements

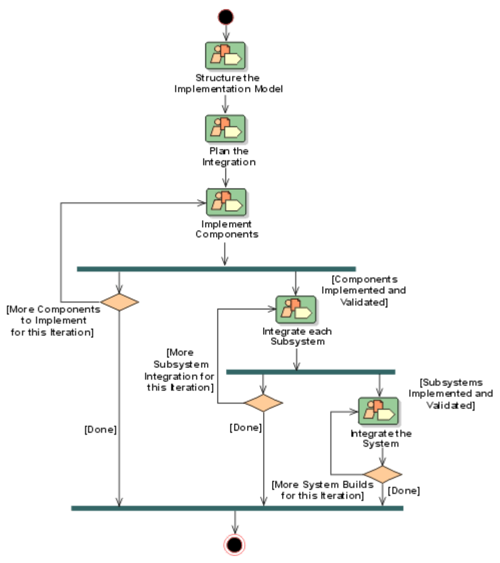
www.lloseng.com

## Project Management



oseng.com

## Implementation



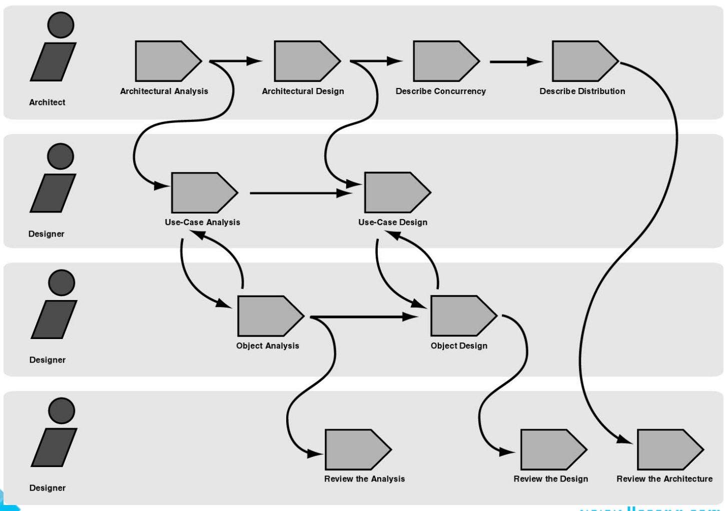
Structure the Implementation Model
Plan the Integration
Implement Components
[Components Implemented and Validated]
[More Components to Implement for this Iteration]
Integrate each Subsystem
[More Subsystem Integration for this Iteration]
[Subsystems Implemented and Validated]
[Done]
[Done]
Integrate the System
[More System Builds for this Iteration]
[Done]

## RUP Tooling

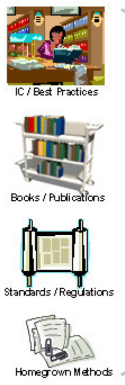
**Describes processes in terms of:**
- **workflows**
- **roles**
- **artifacts**

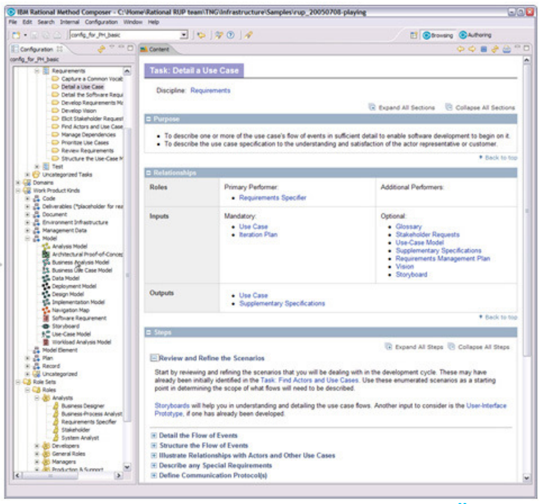
**Provides**
- **templates for deliverables**

## RUP workflow



## Tooling





IBM Rational Method Composer - C:\Home\Rational RUP team\TNG\Infrastructure\Samples\rup_20050708-playing
File Edit Search Internal Configuration Window Help
Configuration
Content
Browsing
Authoring

**Task: Detail a Use Case**

Discipline: Requirements

Expand All Sections  Collapse All Sections

**Purpose**

- To describe one or more of the use case's flow of events in sufficient detail to enable software development to begin on it.
- To describe the use case specification to the understanding and satisfaction of the actor representative or customer.

Back to top

**Relationships**

| | | |
|---|---|---|
| Roles | Primary Performer: • Requirements Specifier | Additional Performers: |
| Inputs | Mandatory: • Use Case • Iteration Plan | Optional: • Glossary • Stakeholder Requests • Use-Case Model • Supplementary Specifications • Requirements Management Plan • Vision • Storyboard |
| Outputs | • Use Case • Supplementary Specifications | |

Back to top

**Steps**

Expand All Steps  Collapse All Steps

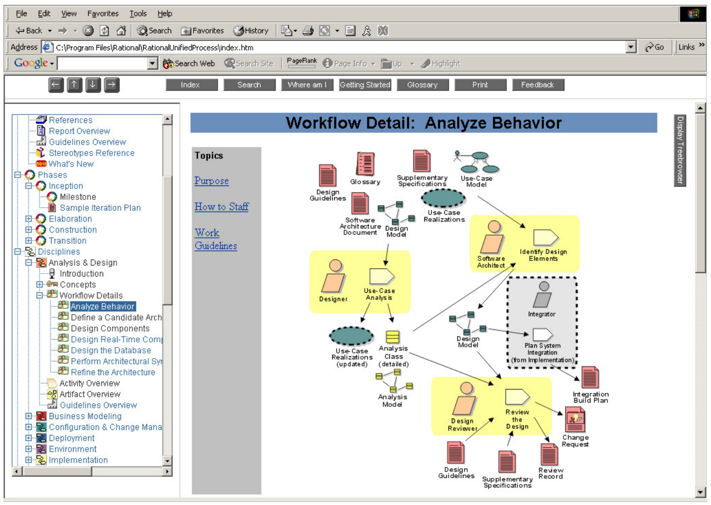
**Review and Refine the Scenarios**

Start by reviewing and refining the scenarios that you will be dealing with in the development cycle. These may have already been initially identified in the Task: Find Actors and Use Cases. Use these enumerated scenarios as a starting point in determining the scope of what flows will need to be described.

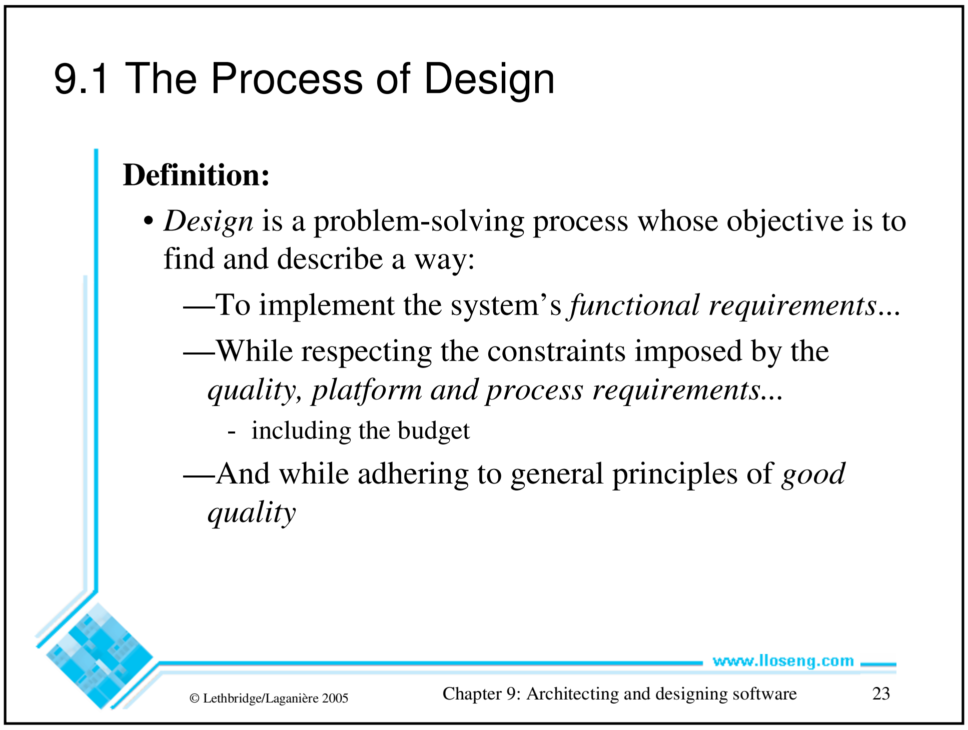
Storyboards will help you in understanding and detailing the use case flows. Another input to consider is the User-Interface Prototype, if one has already been developed.

- Detail the Flow of Events
- Structure the Flow of Events
- Illustrate Relationships with Actors and Other Use Cases
- Describe any Special Requirements
- Define Communication Protocol(s)

## Tooling



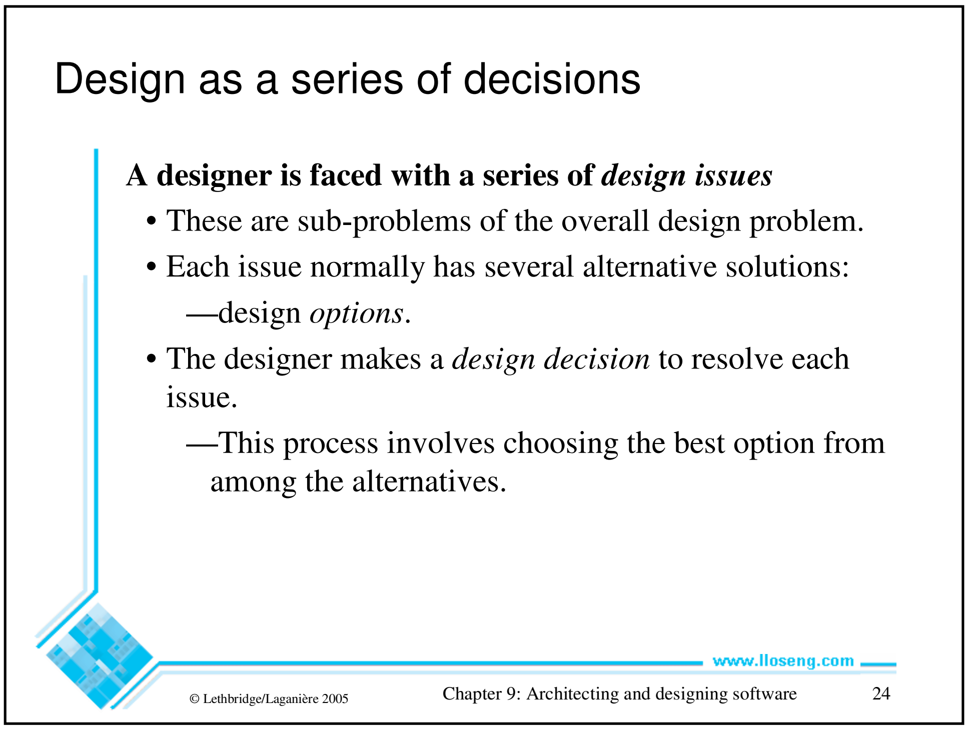
## Design



## 9.1 The Process of Design

**Definition:**

- *Design* is a problem-solving process whose objective is to find and describe a way:
  - —To implement the system's *functional requirements*...
  - —While respecting the constraints imposed by the *quality, platform and process requirements...*
    - including the budget
  - —And while adhering to general principles of *good quality*



## Design as a series of decisions

**A designer is faced with a series of *design issues***

- These are sub-problems of the overall design problem.
- Each issue normally has several alternative solutions:
  - —design *options*.
- The designer makes a *design decision* to resolve each issue.
  - —This process involves choosing the best option from among the alternatives.

## Making decisions

**To make each design decision, the software engineer uses:**

- Knowledge of
    - —the requirements
    - —the design as created so far
    - —the technology available
    - —software design principles and 'best practices'
    - —what has worked well in the past

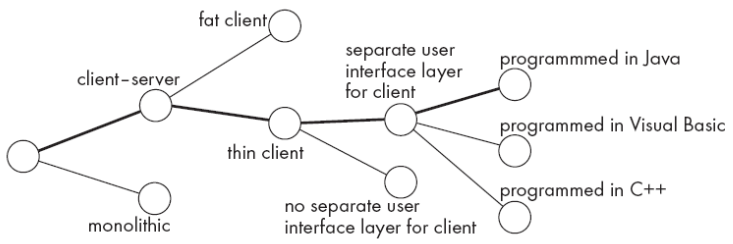
## Document decisions

**- Record the decision**
**- Record the motivation**
**- Record rejected alternatives**

## Design space

**The space of possible designs that could be achieved by choosing different sets of alternatives is often called the *design space***

- For example:



fat client
client-server
separate user interface layer for client
programmmed in Java
thin client
programmed in Visual Basic
programmed in C++
monolithic
no separate user interface layer for client

## Features

According to

FODA: A prominent and user-visible aspect, quality or characteristic of a system.

ODM: A distinguishable characteristic of a system that is relevant to a stakeholder of the system

| In mobile telephones: | In cars: |
|---|---|
| - polyphonic ringtones | - airco |
| - SMS, MMS | - power-steering |
| - dual, tri-band, | - remote key-lock |
| - ... | - ... |

## Feature models

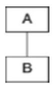
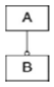
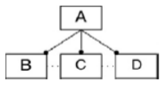
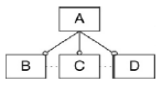
Types of features

**Mandatory**: All systems must have it
e.g. A car must have an engine

**Alternative**:
A system must have one out of multiple options
e.g. Transmission may be manual or automatic

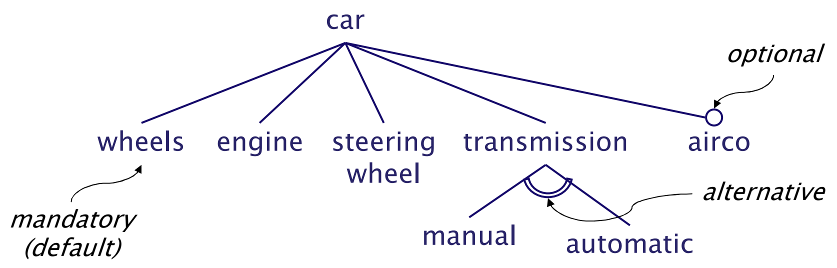
**Optional**: A system may have a feature
e.g. A car may have air-conditioning

Table 1
Explanation of feature diagram elements

| Feature type | Graphical representation |
|---|---|
| **Mandatory**<br>Mandatory feature B has to be included if its parent feature A is selected |  |
| **Optional**<br>Optional feature B may be included if its parent feature A is selected. |  |
| **Alternative**<br>Alternative features are organized in *alternative groups*. Exactly one feature of such a group B, C, D has to be selected if the group's parent feature A is selected. |  |
| **Or**<br>Or features are organized in *or groups*. At least one feature of such a group B, C, D has to be selected if the group's parent feature A is selected. |  |

## Feature Diagram

A hierarchical decomposition of features.
A concept higher in the tree *consists of* its children



Additional annotations that may be used in the feature diagram:
– mutually exclusive features
– rationale for chosing between alternatives
– composition rules: airco may be used if horsepower>100

## Feature Solution Diagrams

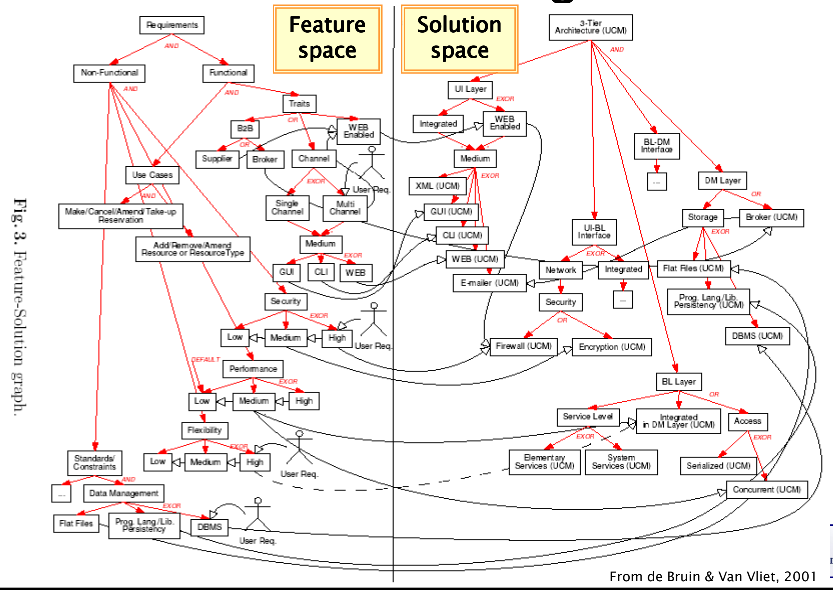


Fig. 3. Feature-Solution graph.

From de Bruin & Van Vliet, 2001

## Different aspects of design

- *Architecture design*:
    - —The division into subsystems and components,
        - How these will be connected.
        - How they will interact.
        - *Interface design*
- *Class design*:
    - —The various features of classes.
- *User interface design*
- *Algorithm design*:
    - —The design of computational mechanisms.
- *Protocol design*:
    - —The design of communications protocol.

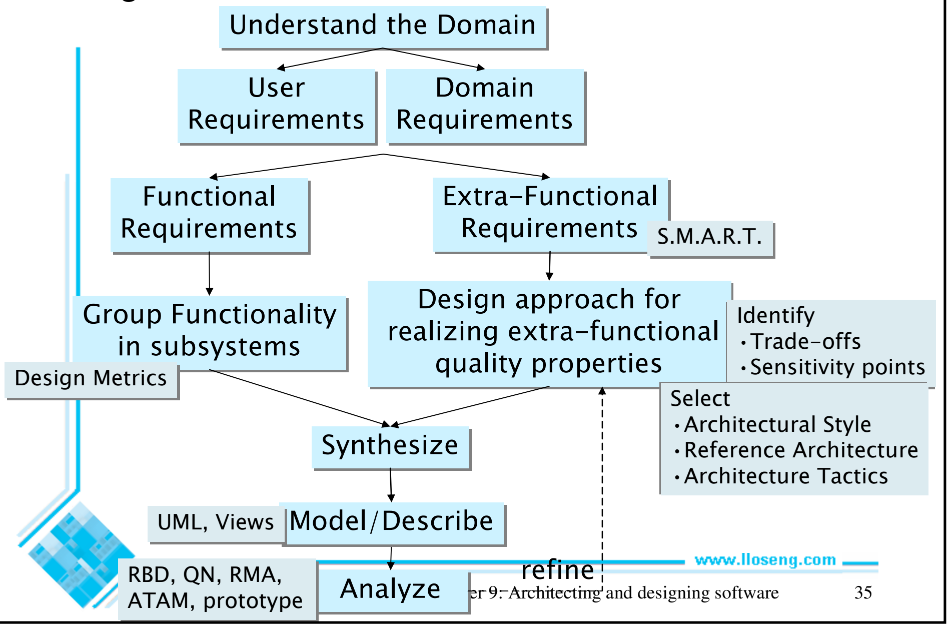
## Architecture is making decisions

The life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in the dark.

**Grady Booch**

- You will not have all information available
- You will make mistakes, but you should learn from them
- There is no objective measure for 'goodness'

## Design of Software Architecture



Understand the Domain
User Requirements
Domain Requirements
Functional Requirements
Extra-Functional Requirements
S.M.A.R.T.
Group Functionality in subsystems
Design Metrics
Design approach for realizing extra-functional quality properties
Identify
• Trade-offs
• Sensitivity points
Select
• Architectural Style
• Reference Architecture
• Architecture Tactics
Synthesize
UML, Views
Model/Describe
RBD, QN, RMA, ATAM, prototype
Analyze
refine