INF3120 –
Utvikling av store programsystemer

Arkitektur

Forelesning 6
14.09.2004
Jan Øyvind Aagedal

Agenda

- Terminology - IEEE
- MDA investigated
- The importance of "why"
- Abstraction
- MDA, metanivåer, UML profiler
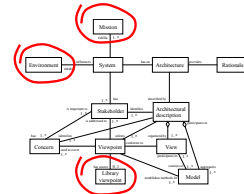- Kvalitetsdimensjoner
- Dokumenteringsråd

Software Architecture

- IEEE Std 1471-2000
  - Recommended Practice for Architectural Description
- Adopted September 2000
- For software-intensive systems
  - Architecture has too long been focussed on hardware-related issues - IEEE strikes back!
- Common frame of reference for architectural descriptions
  - Common terminology
    - architecture, architectural description, model, view, viewpoint, system, stakeholder, concern, …

Motivations

- Two major motivations for explicit architecture
  - Change and Complexity

- Complexity
  - SW systems become complex
  - The context becomes complex
    - Many usage scenarios
  - How to convey
    - Internal structure?
    - Applicability in different contexts?
- Change
  - Panta rei
    - Requirements, underlying platform, competitors, market, ...

- Change ctd.
  - Changes should have limited effects
    - Do not want to have unknown ripple effects
  - Should define the envelope of change
    - What is allowed to change without major consequences
  - Good, old SW engineering principles still apply!
    - Loose coupling
    - High cohesion
 - Flexibility as a competitive advantage
    - AT&T vs Sprint

Architecture of what?

Bus2　Bus3
Bus1　Bus4　　Virtual enterprise

Decomposition

Enterprise architecture →

Actor1　Actor2
SW syst1　SW syst2　　Business

Decomposition

Software architecture →

Comp2　Comp3
Comp1　Comp4　　Software system

Decomposition

Object2　Object3
Object1　Object4　　Software component

Decomposition

Datatype2　Operation1
Datatype1　Datatype3　　Software object

Mission

fulfills　1..*

Env...　...cture.　Rationale

A use or operation for which a system is intended by one or more stakeholders to meet some set of objectives.

Deter...

A viewp... definit... elsewh... archite...

specif... functi...

D... esta... consisting of views expres... architectural description.

...ation of ...ationships to each other and to the environment, and the principles guiding its design and evolution.

...ng and ...or template from which to develop individual views.

the s...

Conceptual Model

IEEE Definitions

- ■ Architecture
  - ■ The fundamental organisation of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.
- ■ Architecture description
  - ■ A collection of products to document an architecture.
- ■ Concern
  - ■ Those interests which pertain the system's development, operation and other aspects that are critical or otherwise important to one or more stakeholders.

IEEE Definitions

- **Environment**
  - Determines the boundaries that define the scope of the system of interest relative to other systems.
- **Library viewpoint**
  - A viewpoint with definition originated elsewhere than in the architectural description.
- **Mission**
  - A use or operation for which a system is intended by one or more stakeholders to meet some set of objectives.

IEEE Definitions
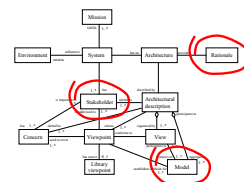
- **Model**
  - Developed using the methods established by its viewpoint, consisting of views expressing an architectural description.
- **Rationale**
  - The Rationale of the architecture.
- **Stakeholder**
  - Has interest in, or concerns relative to the system.

IEEE Definitions
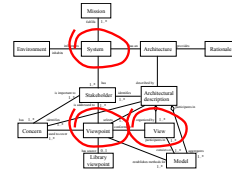
- **System**
    - A collection of components organized to accomplish a specific function or set of functions.
- **View**
    - The expression of a systems architecture with respect to a particular viewpoint. Addresses one or more of the concerns of the system stakeholder.
- **Viewpoint**
    - A specification of the conventions of constructing and using a view. A pattern or template from which to develop individual views.

Defines Evolutionary Envelope

- **Anticipates likely system changes**
    - Other changes than the anticipated ones are often very expensive and uncertain
- **D'Souza & Wills:** "Architecture - The set of design decisions about any system (or smaller component) that keeps its implementors and maintainers from exercising needless creativity."
- **What matters in your system?**
    - You may want to apply relevant patterns to support your concerns

Built for speed:

Built for extensibility:

Mediator

Architectural Description
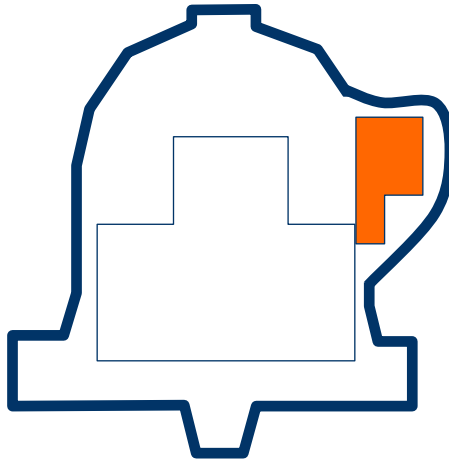
From Bran Selic at Summer
School on Software
Architecture, Turku, Finland,
August 2001

Why?

- Understand the reason, the rationale, the "why" of having the system you are about to develop. This is the only way that will lead you towards the ability to discriminate between important and not-so-important topics.
- The "why" spreads over the design. Each part of it has to have a reason for being the way it is.
- Do not underestimate the "power of why". You should be able to say, at any stage of development, why did you develop it the way you did.
- It is easy to do something just somehow - far more difficult to explain the reasons for the design decision or to justify them
- If you are not able to answer to your inner "why", do not start to implement.
- You may trial a certain technical solution - it provides feedback for the design. Try not to make it the basis of your design, however.

From Teemu Vaskivuo, VTT Electronics

Why in SW architecture

- Software architecture is a discipline trying to advance good practices in software development.

- Good practices are such that help in creating good software

- What is good software like? (functional, fast, secure, available, reusable…?)
- -> Software has more quality dimensions than just one "good or bad"
    - Reliability
    - Performance
    - Security
    - Reusability
    - Availability
    - Integrability etc…

- Some of the dimensions can be improved only at some others' expense
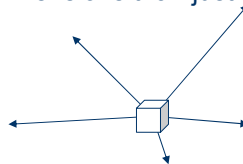
From Teemu Vaskivuo, VTT Electronics

SW architecture, what is it?

- All software has an architecture
- The architecture may be hidden or might be even unknown. Its final (and only) complete instance is the compiled binary file running on its dedicated HW.
- A person cannot understand the architecture from the binary file or by investigating the runtime instance of it. **Abstraction** is required.
- **Abstraction** requires human intuition, it is difficult (nearly impossible) to be automated effectively.
    - The designer is the best person to present an abstraction of his/her design.
    - An expert is the best person to present an abstraction of the area of his/her expertise.
    - SW developer has to often stretch to the both roles.
    - How to make a distinction between important and unimportant details?
        - Experience, practices
        - Try not to mix different levels of abstraction, if possible

From Teemu Vaskivuo, VTT Electronics

Abstraction

- A piece of software is a collection of abstract structures. The amount of structures is practically infinite (since there are no limits for performing abstraction)
    - It depends on the person creating the abstraction, how many structures he wants to represent.
    - It depends on the application, how many structures are needed for a comprehensive design.
- Different structures:
    - data structures
    - structures of timely behavior
    - interaction structures
    - concurrency structures
    - transaction structures
    - component structure
    - ….
- None of the structures is THE Architecture. They all try to represent a different view on it.
- Run-time structures are not present until the software is executed, and they usually exceed human understanding at lower levels, even in information processing sense
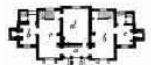
Design Practices

- Tools: Pen and paper, drawing tool, dedicated design SW all do fine.
- The documentation practice shouldn't bound the design (I cannot draw this -> I cannot do this = wrong thinking).
- Standards (UML etc.) are for a large group to be able to understand each other's work. You don't have to think in UML.
- Use abstraction, create components, minimise their relationships.

Arkitekturer og meta-nivå

- Architecture
  - The fundamental organisation of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

Level:	Architecture hierarchy:	Building analogy:
0	Documented system	New home with documentation
1	System architectural description	Architectural description of new home
2	Methodology for developing system architecture	Rules and conventions for developing the architecture of homes
3	MDA framework	Building laws and regulations

MDA: "Model driven" - a definition

- A system development process is model driven if
  - the development is mainly carried out using conceptual models at different levels of abstraction and using various viewpoints
  - it distinguishes clearly between platform independent and platform specific models
  - models play a fundamental role, not only in the initial development phase, but also in maintenance, reuse and further development
  - models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation
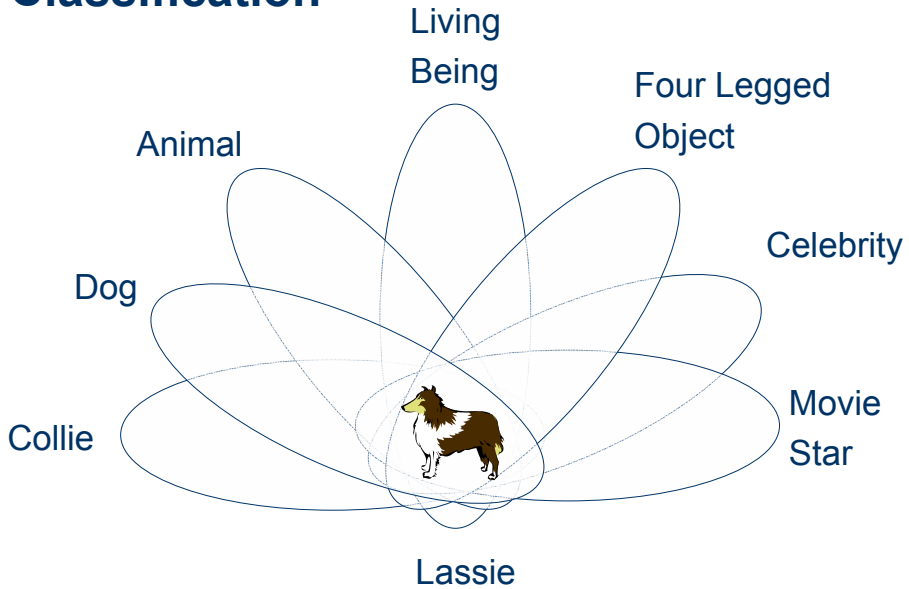
Metamodels

- Metamodels are specifications
  - models are valid if no false statements according to metamodel (i.e., well-formed)
- Metametamodel
  - model of metamodels
  - reflexive metamodel, i.e., expressed using itself
    - ref. Kurt Gödel
  - minimal reflexive metamodel
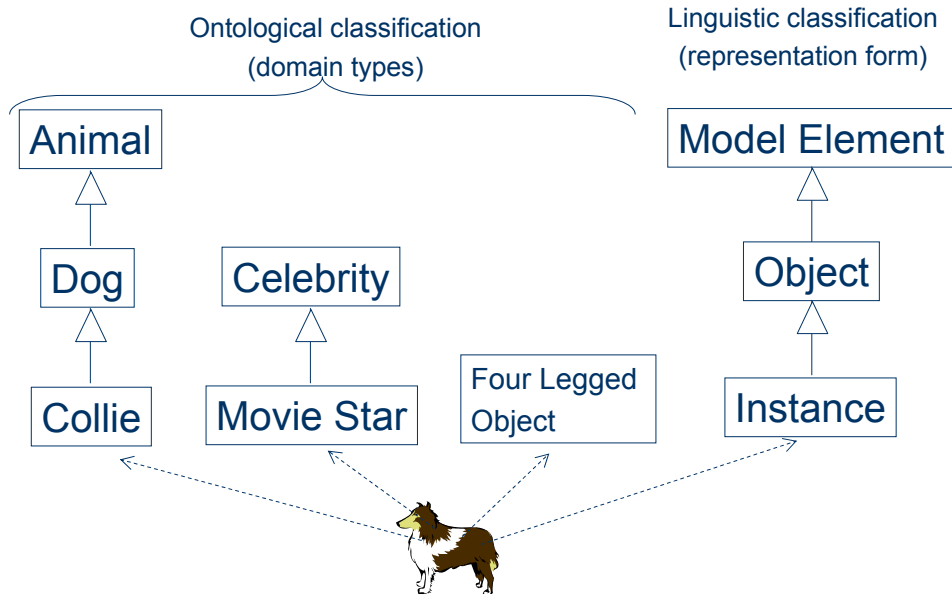   - can be used to express any statement about a model

Meta levels in OMG

- M0 – what is to be modelled (Das Ding an sich)
- M1 – Models (Das Ding für mich)
  - May contain both class/type and instance models
- M2 – Metamodels
- M3 – The metametamodel

- Interpretation (not instantiation!) crosses meta-layers, theories reside in one layer (e.g., instance models can be deduced from class models)
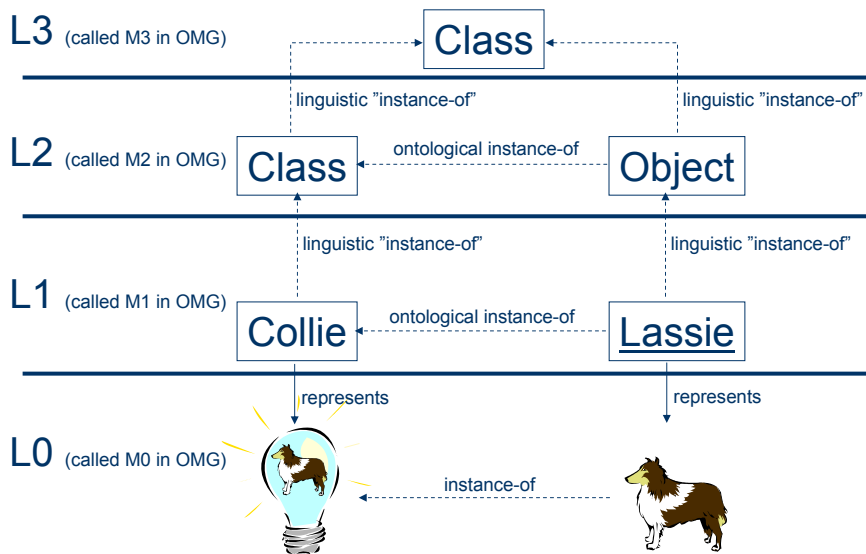
Classification

Living Being

Four Legged Object

Animal

Celebrity

Dog

Collie

Movie Star

Lassie

Classification Dimensions

Ontological classification
(domain types)

Linguistic classification
(representation form)

Animal

Model Element

Dog

Celebrity

Object

Collie

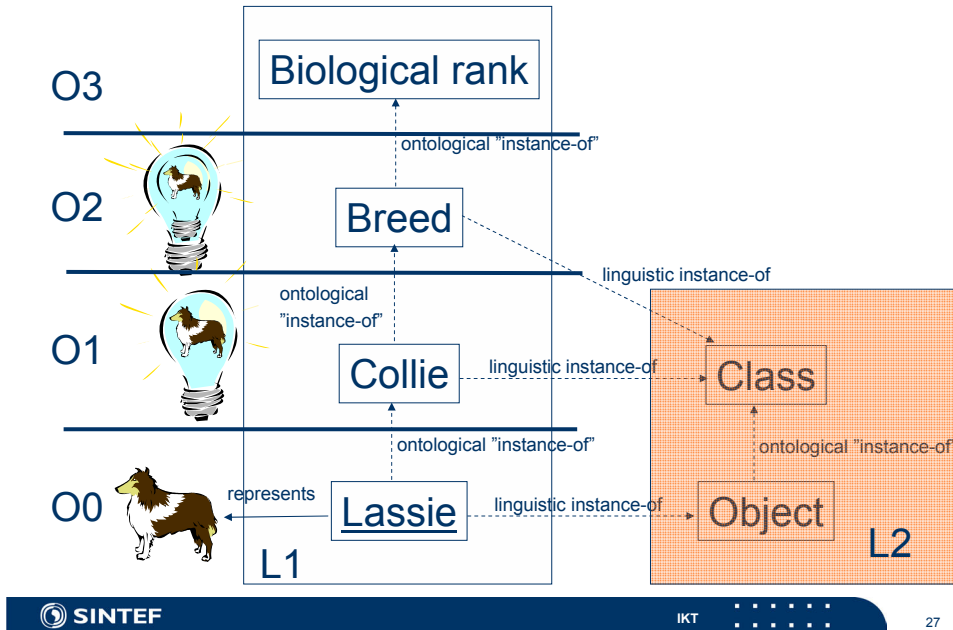Movie Star

Four Legged Object

Instance

Kinds of metamodels

- Two kinds of information of a set of models are modelled in metamodels
  - Form (linguistic aspects)
    - OMG is predominantly occupied with this
  - Content (ontological aspects)

Linguistic metamodelling

L3 (called M3 in OMG)
Class

linguistic "instance-of"   linguistic "instance-of"

L2 (called M2 in OMG)   Class ← ontological instance-of — Object

linguistic "instance-of"   linguistic "instance-of"

L1 (called M1 in OMG)   Collie ← ontological instance-of — Lassie

represents   represents

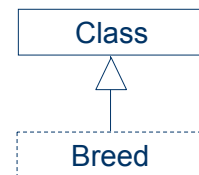L0 (called M0 in OMG)   instance-of
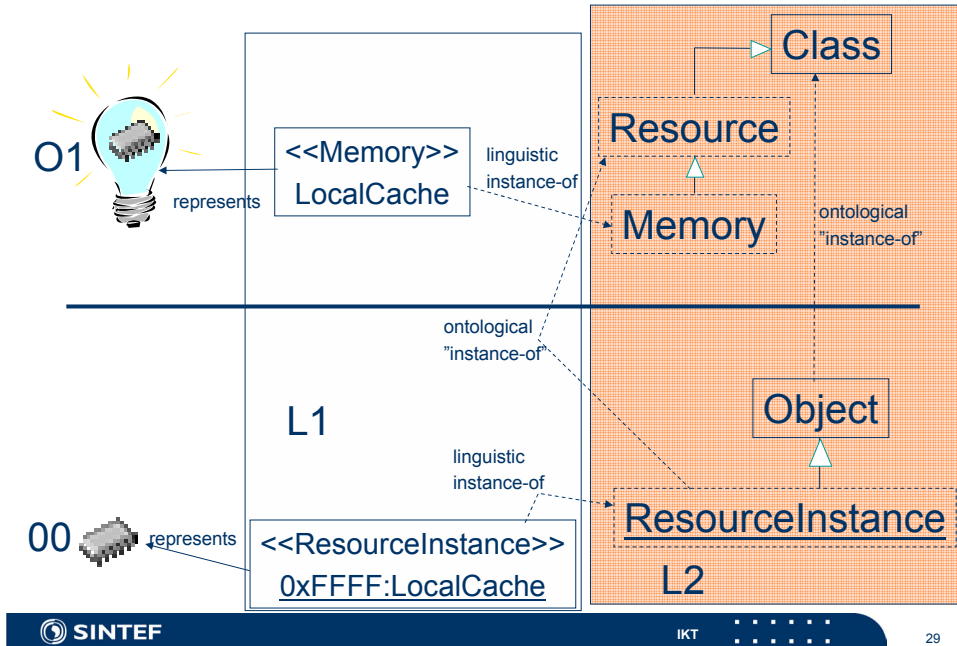
Ontological metamodelling

UML profiles

- Define the meaning of modelling elements
  - interpretation, i.e., relate stereotyped classifiers to real world concepts such as memory, process, resource
- Mostly content (ontological), but must relate to form (linguistic)
- Poor expressional power – stereotypes
  - no relations between O 0/1/2 concepts
  - cast as form, but is really content
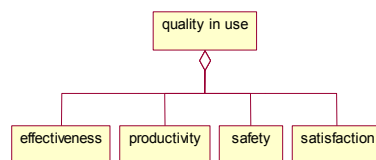
Resource model as a UML profile

O1 represents <<Memory>> LocalCache — linguistic instance-of

Class

Resource

Memory

ontological "instance-of"

L1

ontological "instance-of"

linguistic instance-of

Object

00 represents <<ResourceInstance>> 0xFFFF:LocalCache

ResourceInstance

L2

Architecture Description Techniques

- ADLs (Architecture Description Languages)
  - Rapide, UniCon, Wright, Aesop, ArTek, SADL, Meta-H, C-2, ...
    - Lexical, research-oriented (not widely used)
  - Lessons learned from the ADL community
    - Connectors, Interfaces
    - Multiple representations
      - Refinement, PIM/PSM
    - Styles
      - Domain-specific vocabulary
      - Impose topology constraints
    - Visualisation
      - Domain-specific visual conventions
- UML (Unified Modeling Language)
  - Graphical
  - Standard, widely used
  - No agreed set of diagrams to specify system architectures

Quality of Service Support in Software Architecture

- Architecture defines evolutionary envelope within which acceptable quality is maintained
  - But what kind of qualities are affected?
- Qualities define the "goodness" of the system (or its architecture)
- Heaps of -ilities
  - Subjective vs. objective quality
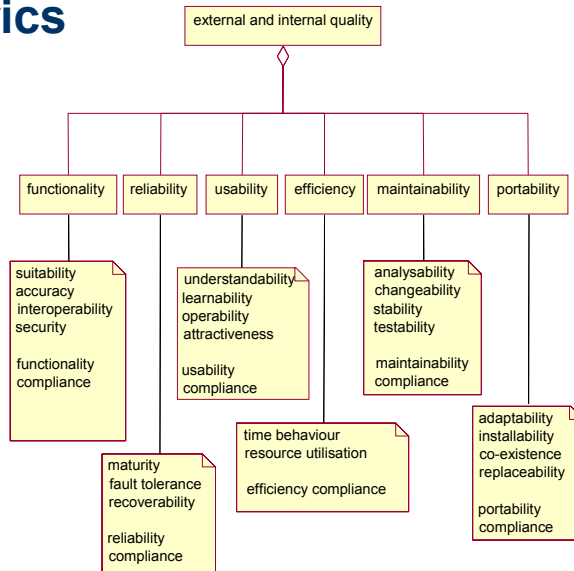 - Design-time vs. run-time qualities

ISO 9126 - Software Product Quality

```
                    quality in use
                         |
     ┌───────────┬───────┴───────┬───────────┐
 effectiveness  productivity   safety    satisfaction
```

- Quality in use - related to a specific context of use
  - effectiveness – the capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use.
  - productivity – the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in the specified context of use.
  - safety – the capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.
  - satisfaction – the capability of the software product to satisfy users in a specified context of use.

ISO 9126 - External and Internal Quality Metrics

■ Irrespective of context of use

```
                          ┌──────────────────────────────┐
                          │ external and internal quality │
                          └──────────────────────────────┘
```

functionality	reliability	usability	efficiency	maintainability	portability

suitability
accuracy
interoperability
security

functionality
compliance

maturity
fault tolerance
recoverability

reliability
compliance

understandability
learnability
operability
attractiveness

usability
compliance

time behaviour
resource utilisation

efficiency compliance

analysability
changeability
stability
testability

maintainability
compliance

adaptability
installability
co-existence
replaceability

portability
compliance

Functionality

■ The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

■ **Suitability**
  - The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.

■ **Accuracy**
  - The capability of the software product to provide the right or agreed results or effects with the needed degree of precision.

■ **Interoperability**
  - The capability of the software product to interact with one or more specified systems.

■ **Security**
  - The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them

■ **Functionality compliance**
  - The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality.

Reliability

- The capability of the software product to maintain specified level of performance when used under specified conditions.

- Maturity
  - The capability of the software product to avoid failure as a result of faults in the software.
- Fault tolerance
  - The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

- Recoverability
  - The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.
- Reliability compliance
  - The capability of the software product to adhere to standards, conventions or regulations relating to reliability.

Usability

- The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.

- Understandability
  - The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
- Learnability
  - The capability of the software product to enable the user to learn its application.

- Operability
  - The capability of the software product to enable the user to operate and control it.
- Attractiveness
  - The capability of the software product to be attractive to the user.
- Usability compliance
  - The capability of the software product to adhere to standards, conventions, style guides or regulations relating to usability.

Efficiency

- The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.

- Time behaviour
  - The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.

- Resource utilisation
  - The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.

- Efficiency compliance
  - The capability of the software product to adhere to standards or conventions relating to efficiency.

Maintainability

- The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.

- Analysability
  - The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.

- Changeability
  - The capability of the software product to enable a specified modification to be implemented.

- Stability
  - The capability of the software product to avoid unexpected effects from modifications of the software.

- Testability
  - The capability of the software product to enable modified software to be validated.

- Maintainability compliance
  - The capability of the software product to adhere to standards or conventions relating to usability.

Portability

- The capability of the software product to be transferred from one environment to another.

- Adaptability
  - The capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered.
- Installability
  - The capability of the software product to be installed in a specified environment.

- Co-existence
  - The capability of the software product to co-exist with other independent software in a common environment sharing common resources.
- Replaceability
  - The capability of the software product to be used in place of another specified software product for the same purpose in the same environment.
- Portability compliance
  - The capability of the software product to adhere to standards or conventions relating to portability.

Development-time qualities (from Catalysis)

- Affected by architecture:
  - Modifiability- Can the system be modified efficiently?
    - E.g., low coupling and high cohesion
  - Reusability - Are there units in the system that can are candidates for use elsewhere?
    - E.g., does the system use standards?
  - Portability - The ability to change platform
    - E.g., layering
  - Buildability - Is it easy to implement?
    - E.g., use existing frameworks
  - Testability - Can one easily define test scenarios?
    - E.g., precise requirement specifications

Runtime qualities (from Catalysis)

- Affected by architecture:
  - Functionality - does the system assist users in their tasks?
  - Usability - is it intuitive to use for all users?
  - Performance - does it perform adequately when running?
    - E.g., response time, transaction volume, ...
  - Security - does it prevent unauthorised access?
  - Reliability and availability - is it available and correct over time?
  - Scalability - can it cater for increased volume?
  - Upgradability - can it be upgraded at runtime?

- Single key quality: Conceptual integrity of an architecture

Quality of Service (QoS)

- QoS is a general term that covers system performance, rather than system operation (i.e., functionality)
- Extra-functional properties
  - degrees of satisfaction as opposed to satisfied / not satisfied
- Examples:
  - availability, reliability, precision, fault-tolerance, capacity, throughput, delay, …
- Most common for multimedia, command and control, simulations, distributed systems, …
- Less common for information systems
  - there are needs! (transaction-based, security aware, etc.)

Seven Principles of Sound Documentation (from Paul Clements, SEI, CMU)

- Certain principles apply to all documentation, not just documentation for software architectures.

- 1. Write from the point of view of the reader.
- 2. Avoid unnecessary repetition.
- 3. Avoid ambiguity.
- 4. Use a standard organization.
- 5. Record rationale.
- 6. Keep documentation current but not too current.
- 7. Review documentation for fitness of purpose.
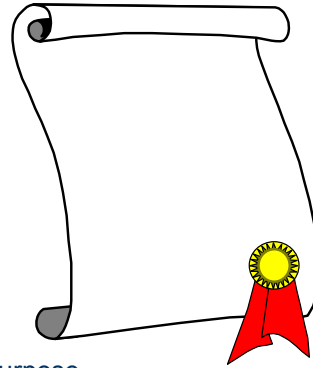
1. Write from the point of view of the reader.

- What will the reader want to know when reading a document?
  - Make information easy to find!
  - Your reader will appreciate your effort and be more likely to read your document.

- Signs of documentation written for the writer's convenience:
  - stream of consciousness: the order is that in which things occurred to the writer
  - stream of execution: the order is that in which things occur in the computer

2. Avoid unnecessary repetition.

- Each kind of information should be recorded in exactly one place.

- This makes documents easier to use and easier to change.

- Repetition often confuses, because the information is repeated in slightly different ways. Which is correct?

- Question: When is repetition OK?

3. Avoid ambiguity.

- Documentation is for communicating information and ideas. If the reader misunderstands, the documentation has failed.

- Precisely-defined notations/languages help avoid whole classes of ambiguity.

- If your documentation uses a graphical language
  - always include a key
  - either point to the language's formal definition or give the meaning of each symbol. Don't forget the lines!

3. Avoid ambiguity (cont'd.)

- Box-and-line diagrams are a very common form of architectural notation.
- But what do they mean?
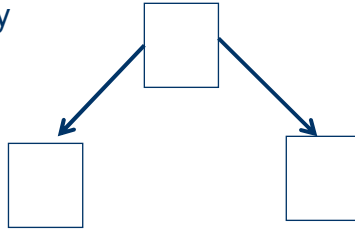
- These do not show an
- architecture, but only the beginning of one.

- If you use one, always define precisely what the boxes are and what the lines are.

- If you see one, ask the owner what it means. The result is usually very entertaining.

4. Use a standard organization.

- Establish it, make sure your documents follow it, and make sure that readers know what it is.

- A standard organization
  - helps the reader navigate and find information

  - helps the writer place information and measure work left to be done

  - embodies completeness rules, and helps check for validation

4. Use a standard organization (cont'd.)

- **Corollaries:**

    - Organize the documentation for ease of reference.
        - A document may be *read* once, if at all.
        - A successful document will be *referred to* hundreds or thousands of times.
        - Make information easy to find.
        - Question: How?

    - Don't leave incomplete sections blank; mark them "to be determined"
        - Question: Why?

5. Record rationale.

- Why did you make certain design decisions the way you did?

- Next week, next year, or next decade, how will you remember? How will the next designer know?

- Recording rationale requires discipline, but saves enormous time in the long run.

- Record rejected alternatives as well.

6. Keep documentation current but not too current.

- Keep it current:
    - Documentation that is incomplete, out of date, does not reflect truth, and does not obey its own rules for form is not used.
    - Documentation that is kept current is used.
    - With current documentation, questions are most efficiently answered by referring the questioner to the documentation.
    - If a question cannot be answered with a document, fix the document and then refer the questioner to it.
    - This sends a powerful message.

6. Keep documentation current but not too current (cont'd.)

- Don't keep it *too* current
    - During the design process, decisions are considered and re-considered with great frequency.
    - Revising the documentation every five minutes will result in unnecessary expense.
    - Choose points in the development plan when documentation is brought up to date
    - Follow a release strategy that makes sense for your project.,

7. Review documentation for fitness of purpose

- Only the intended users of a document can tell you if it
    - contains the right information
    - presents the information in a useful way
    - satisfies their needs

- Plan to review your documents with representatives of the groups for whom it was created.

- *Active design reviews* are a good technique.