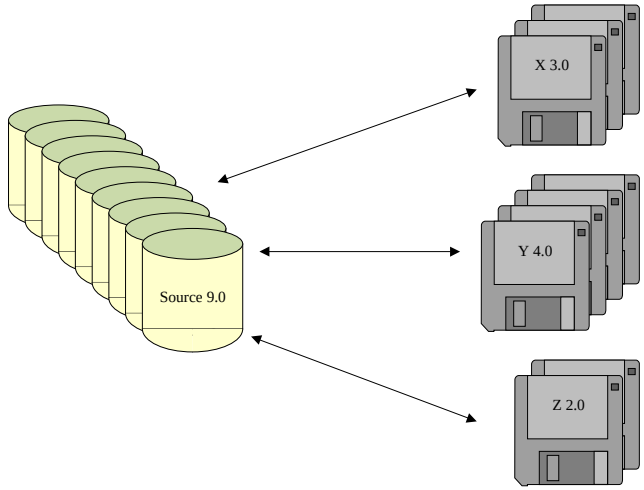
Defining the System

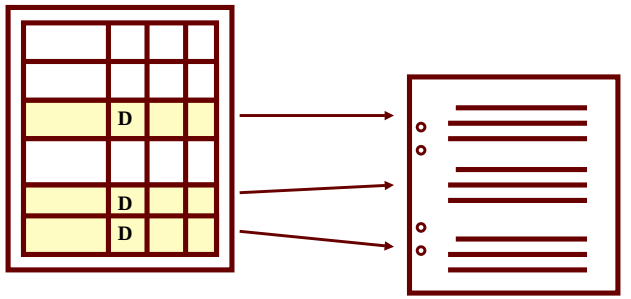
The Products



Product Plan: Feature Statement for all Products

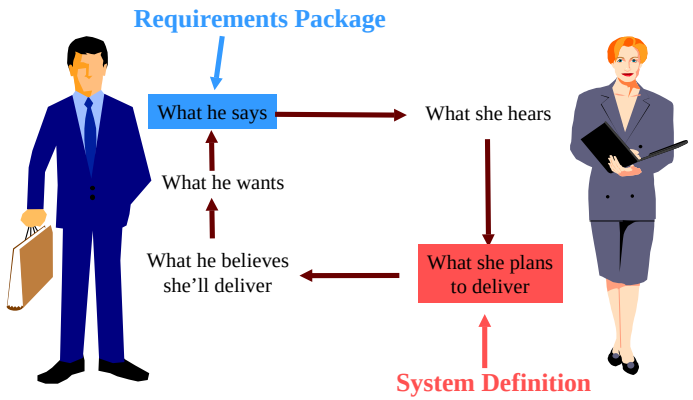
Requirement	From rev	To rev	Valid in
Descriptive text.	A	C	X,Y
….	A	-	Y,Z
…..	D	-	Z
….	D	E	X,Y,Z

Requirement Packages: Requirements towards a Project



Product plan Requirements package

Requirement Validation



Aspects on System Definition

- Physical appearance
- Logical appearance
- System Boundary
- Function
- Non-functional aspects

Physical Appearance (Example)

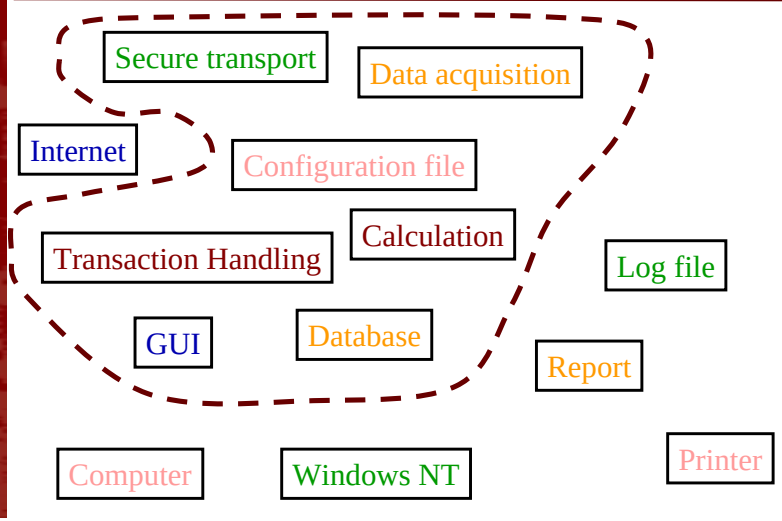
The system consists of a globally distributed network of computers, connected via internet. Two types of computers exists:

- *Presentation stations, with a graphical user interface (GUI).*
- *Acquisition stations without a GUI to which data acquisition equipment is connected.*

Logical Appearance (Example)

- *The operator of a presentation station perceives the system as an explorer type application running under a Windows-like GUI. The explorer view presents all acquisition points in a tree hierarchy. Each leaf in the tree represents a measured data of any of the predefined types. The tree represents an arbitrary logical grouping of the acquisition points, which is transparent with respect to geographical distribution.*

System Boundary: What is part and what is not.



Visibility

- White box view
  - Invisible for the user.
- Gray box view
  - Perceived by the user
- Black box view
  - Accessible by the user.

Function

- What does the system do?
- Ex:
  - Data acquisition
  - Data presentation
  - Network management
  - Database handling
  - …..

Non-Functional Aspects

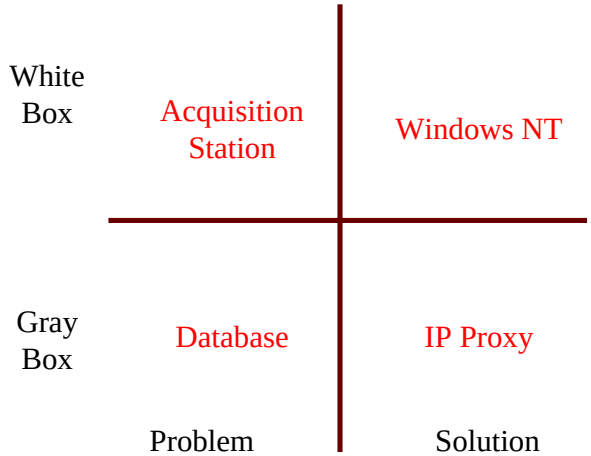
- Performance
- Quality
- Modularity
- Documentation
- Support
- Power consumption
- Environment
- Color
- …..

Domain Model

- Concepts, principles, patterns
- Problem domain: Valid for all systems that fulfills the requirements
- Solution domain: Valid for the way this system fulfills the requirements.
- White and gray box view

Domain models are highly reusable assets!

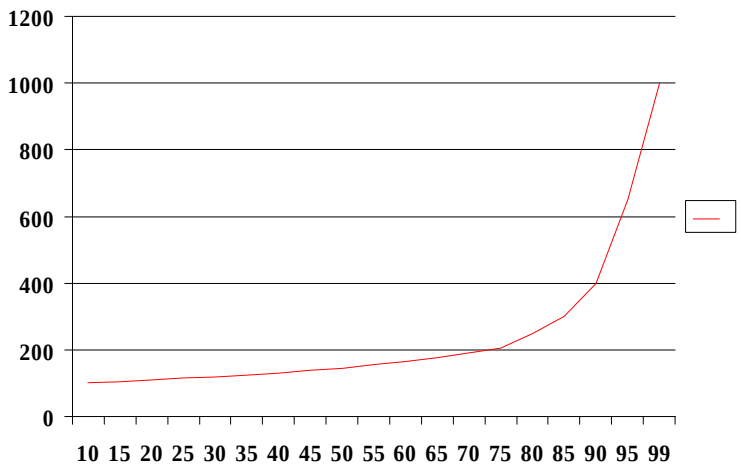
Domain Concepts (Example)



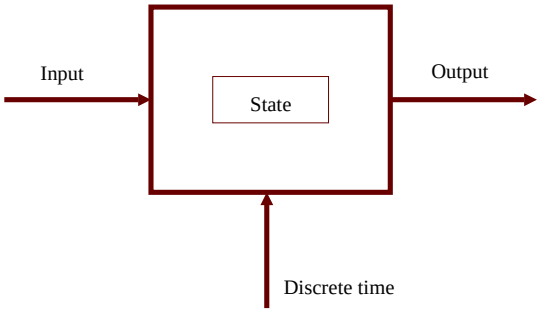
Formally Definable Systems?

- Compiler?
- Weather Prediction System?
- Telephony Exchange?

Cost of Formality



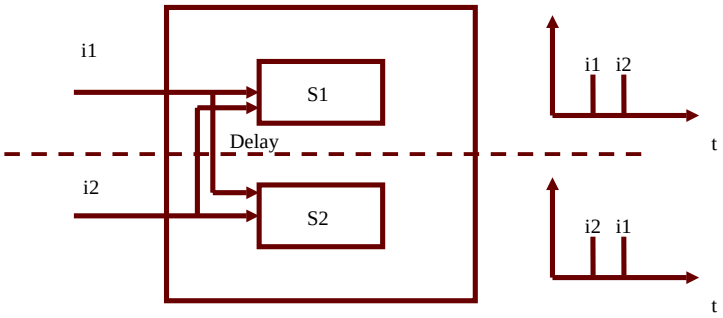
The System as an Automata



Automata Definition Effort

Number of Significant States
X
Number of Significant Input Values

Asynchronous State



Applicability of Automata Model

- Is there a synchronous state?
- Is transition time < time step?
- Is number of states finite?
- Is value range of input finite?
- Is behaviour fixed?

Compiler

- Is there a synchronous state? Yes
- Is transition time < time step? Yes*
- Is number of states finite? Yes
- Is value range of input finite? Yes
- Is behaviour fixed? Yes

*) Batch Processing: Internally generated time step.

Weather Prediction System

- Is there a synchronous state? Yes
- Is transition time < time step? Yes*
- Is number of states finite? No
- Is value range of input finite? No
- Is behaviour fixed? ?

*) Batch Processing: Internally generated time step.

Telephony Exchange

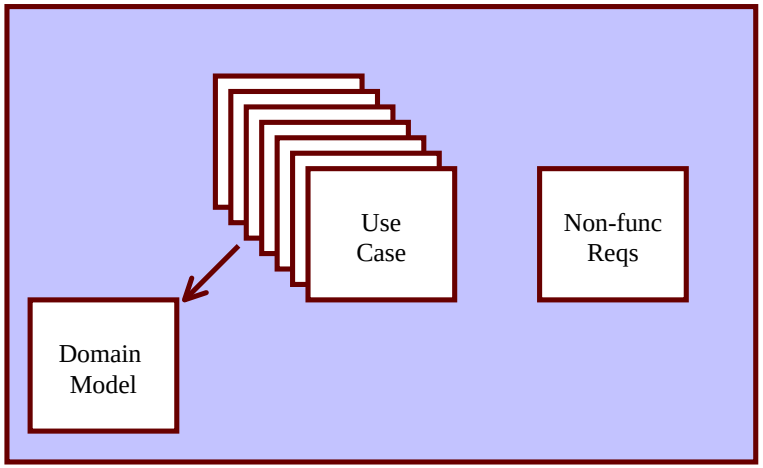
- Is there a synchronous state? No
- Is transition time < time step? No
- Is number of states finite? Yes*
- Is value range of input finite? Yes
- Is behaviour fixed? Yes

*) For practical purposes

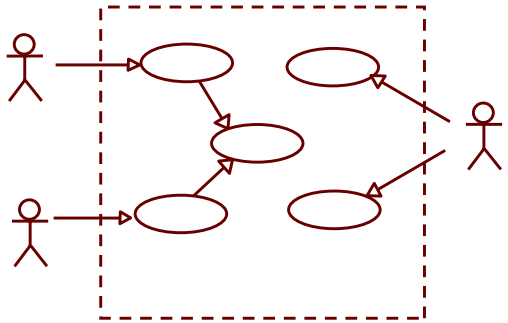
Event Driven Modelling

- Asynchronous state: Decomposition into multiple synchronous states.
- Transition time > time step: Decomposition of system transition into multiple internal transitions.
- SDL, UML (Harel State Charts)...
- Assumption: Memory is unlimited.
- Number of states in practice very high.
- Pseudo formalism!

Defining the System by Use Case Modeling



Use Case Model



- Defines System Boundary
- Identifies Functionality

Use Case Description

- Word processor document
- 2-4 pages
- Unique name
- Narrative description (1 paragraph)
- List of significant conditions
- List of significant influences
- Scenario descriptions considering conditions and influence.
  - Precondition
  - List of interactions
  - Extension points
  - Postcondition

Scenarios

- A flow of event under a certain combination of condition and influences.
- No choice points.
- Each use case is described by a sufficient number of scenarios.

Sufficient?

- Internal state composed of i internal boolean conditions.
- External influence composed of e external boolean conditions
- Number of possible scenarios = 2^{i+e}
- If transitions are non-atomic, also consider that the state may be altered by other use cases during the execution of this use case.
  Number of possible scenarios >> 2^{i+e}
- Sufficient: One normal + one per not-normal condition and influence => 1 + i + e scenarios.

Example:

- Use Case "Save File As…"
  - Scenario 1: "Successful save"
  - Scenario 2: "Disk full"
  - Scenario 3: "Invalid file name given"
  - Scenario "Invalid file name given when hard disk full" not defined.
- Incomplete definition resolved by designer decision during design phase.

How to design working systems from incomplete definitions?

Use humans!

Humans have the necessary skill.
This is why software is developed by humans.

Creating software from formal specifications is the task for compilers.

Creating software that has reasonable behavior in undefined situations is the task for humans.