Lecture 2: Basics of RE

Last Week:
INTRO
Syllabus
Course Goals
Literature on RE

This Week:
Basics of RE
RE in the lifecycle
Types of problem domain
Processes, methods & techniques
A little systems theory

Next Week:
Elicitation (I)
Traditional approaches
Interviews & Questionnaires
Prototyping

Importance of RE: background

→ Problems
- Increased reliance on software
  - E.g. cars, dishwashers, cell phones, web services, …
- Software now the biggest cost element for mission critical systems
  - E.g. Boeing 777
- Wastage on failed projects
  - E.g. 1997 GAO report: $145 billion over 6 years on software that was never delivered
- High consequences of failure
  - E.g. Ariane 5: $500 million payload
  - E.g. Intel Pentium bug: $475 million

→ Key factors:
- Certification costs
  - E.g. Boeing 777: >40% of software budget spent on testing
- Re-work from defect removal
  - E.g. Motorola: 60-80% of software budget (was) spent on re-work
- Changing Requirements
  - E.g. California DMV system

Solutions??

→ No "Silver Bullet" *(Brooks)*
- Software is complex for its size
- Software is invisible and abstract
- No fabrication step; hence software is "modifiable"(!)

→ But: Early modeling and analysis is important
- Defects are cheaper to remove the earlier they are found (Boehm)
- Requirements defects are more likely to be safety-related (Lutz)
  - E.g. Voyager and Galileo

→ Early modeling and analysis is not enough
- Need to communicate requirements to everyone
- Need to seek agreement from all stakeholders
- Need to understand the context for the system
- Need to understand the context for the development process
- Need to keep up to date as the requirements evolve

Refs: *Brooks 1987, Boehm 1981, Lutz 1993*

Basic Definitions

→ What is a Requirement?
- Something that someone needs to solve a problem or achieve an objective:
  - *"A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component".* [IEEE Std]

→ What is Requirements Engineering?

"…Requirements Engineering is the branch of systems engineering concerned with real-world goals for, services provided by, and constraints on software systems. Requirements Engineering is also concerned with the relationship of these factors to precise specifications of system behaviour and to their evolution over time and across system families…" [Zave94]

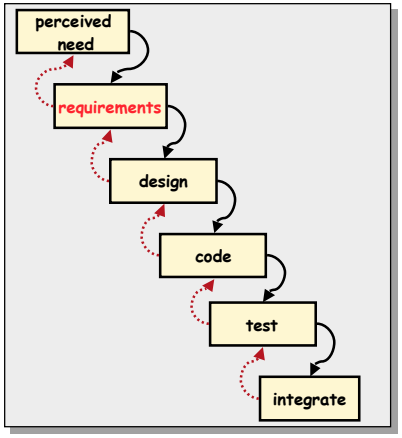
"… RE is concerned with identifying the purpose of a software system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies." [RE'01 CfP]

RE vs Systems Analysis

→ RE grew out of systems analysis:
- Systems analysis focuses on information systems within an organization
- Has developed or adopted mostly informal notations, tools and methodologies
  - E.g. DFDs, E-R, OO diagrams,…
- Widely practiced, largely through management consulting companies
- Taught at undergraduate and graduate level in Management Studies and, increasingly, Industrial Engineering and Computer Science programmes

→ RE goes beyond systems analysis:
- Encompasses the entire formalization problem:
  - From "a business need" to "a precise specification"
- Expands the scope beyond information systems:
  - real-time systems
  - embedded systems
  - interactive applications
  - component-based software
  - web services
- *(But perhaps less emphasis on management issues & business processes)*

Waterfall Model



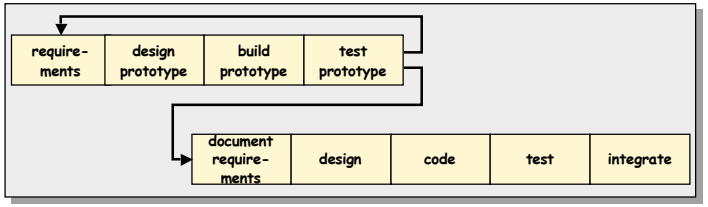
→ View of development:
- a process of stepwise refinement
- largely a high level management view

→ Problems:
- Takes a static view of requirements, ignores volatility
- Lack of user involvement once specification is written
- Unrealistic separation of specification from design
- Doesn't accommodate prototyping, reuse, etc.

Source: Adapted from Dorfman, 1997, p7 & Loucopoulos & Karakostas, 1995, p29

Prototyping lifecycle

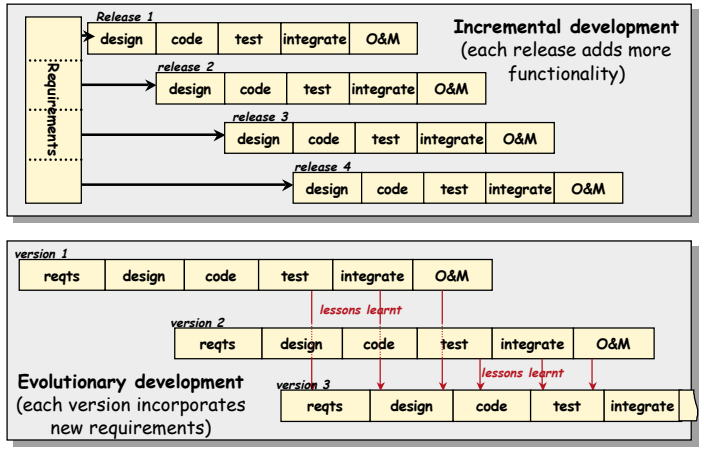
Source: Adapted from Dorfman, 1997, p9



→ Prototyping is used for:
- understanding the requirements for the user interface
- examining feasibility of a proposed design approach
- exploring system performance issues

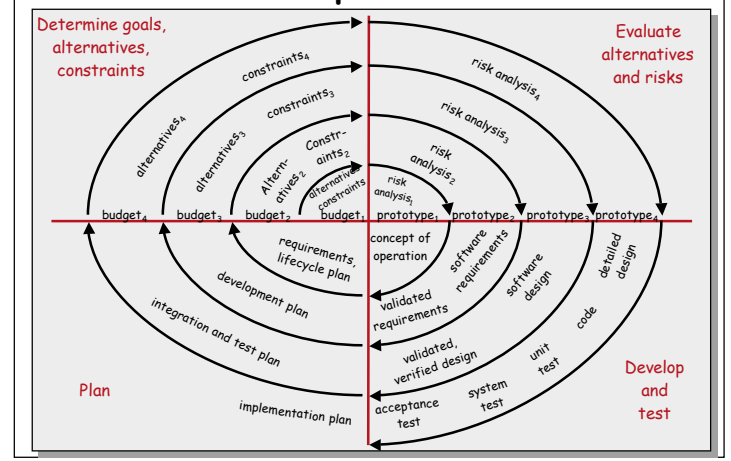
→ Problems:
- users treat the prototype as the solution
- a prototype is only a partial specification

Phased Lifecycle Models



Source: Adapted from Dorfman, 1997, p10

The Spiral Model



Source: Adapted from Pfleeger, 1998, p57

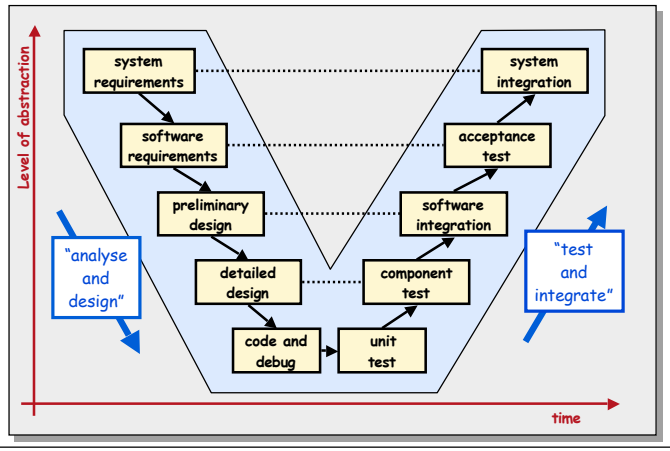
Requirements in the Spiral Model

- → Spiral model is a risk management model
- → For each iteration:
  - plan next phases;
  - determine objectives & constraints;
  - evaluate alternatives;
  - resolve risks;
  - develop product
- → Includes as Requirements processes:
  - Requirements risk analysis (using simulation and prototyping)
  - Planning for design

  (these reduce the risk that requirements process has to be repeated because requirements cannot be met)
- → Problems:
  - Spiral model cannot cope with unforeseen changes during development
    - e.g. emergence of new business objectives

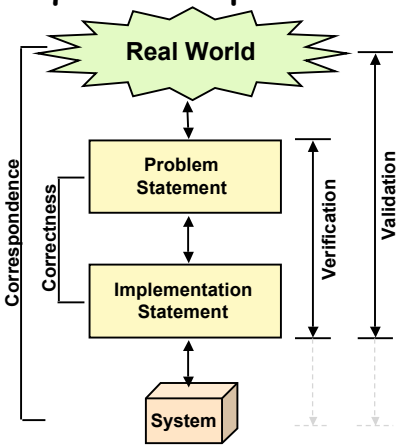
Source: Adapted from Loucopoulos & Karakostas, 1995, p30

V-Model



The essential requirements process

- → Understand the problem
  - elicitation, requirements acquisition, etc.
- → Formally describe the problem
  - specification, modelling, etc.
- → Attain agreement on the nature of the problem
  - validation, conflict resolution, negotiation
  - requirements management - maintain the agreement!



Source: Adapted from Loucopoulos & Karakostas, 1995, p20 and Blum, 1992

Types of RE Problem Domain

- Normal or Revolutionary design?
  - Normal design: old problems, whose solutions are well known
    - engineering codifies standard solutions
    - engineer selects appropriate methods and technologies
  - Revolutionary: never been done, or past solutions have failed
    - This is not really in the realm of "engineering"!
- Type of software needed:
  - Static or Dynamic?
    - Static: all input data available before processing starts
    - Dynamic: data continues to arrive during processing
  - Sequential or Parallel?
  - Data- Control- or Algorithm- hard?
    - Data-hard: complex data moves across system boundary
    - Control-hard: complex control laws describe how the system should control its environment (or vice versa)
    - Algorithm-hard: the computations performed by the system are complex
  - Deterministic or Non-deterministic?

Source: Adapted from Davis, 1990, p30

Types of RE project

- Source of Requirements:
  - Customer-driven
    - involve a specific customer who needs a system to solve a specific problem
  - Market-driven
    - involve a developer who needs to develop a system to be sold in the market
  - Hybrid
    - developed for a specific customer, but want to market the software eventually
- Nature of the Product
  - One-off ('bespoke') vs. Packaged ('shrink wrapped')
  - Single system vs. Product Family ('product line')
  - New system vs. Upgrade to existing system
- These questions affect the role of Requirements:
  - as a statement of the problem to be solved
  - as a contract between customer and developers
  - for communication between designer, customer and end-users
  - to support system evolution
  - to support design validation

Software Types

- Information Systems
  - software to support organizational work
  - includes files/databases as well as applications
  - More than 70% of all software falls in this category, written in languages such as COBOL, RPG and 4GLs.
    - Examples: Payroll, Employee Records, Accounts payable/receivable, Customer records, Transaction records
- Embedded Systems
  - software that drives some sort of a hardware process
    - Examples: industrial plant, an elevator system, or a credit card machine.
- Generic Services
  - systems that provide some form of generic service
    - Examples: many internet applications, e.g. search engines, stock quote services, credit card processing, etc.
  - Such systems will be developed using a variety of languages and middleware, including Java, C++, CORBA, HTML/XML etc.

Processes, Methods, Techniques...

A *notation* is a representation scheme (or language) for expressing things; e.g., Z, first order logic, dataflow diagrams, UML.

A *technique* prescribes how to perform a particular (technical) activity - and, if necessary, how to describe a product of that activity in a particular notation; e.g, use case diagramming,

A *method* provides a technical prescription for how to perform a collection of activities, focusing on integration of techniques and guidance about their use; e.g., SADT, OMT, JSD, KAOS, RUP(?).

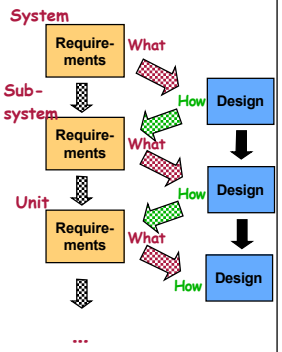
A *Process model* is an abstract description of how to conduct a collection of activities, focusing on resource usage and dependencies between activities.

A *Process* is an enactment of a process model, describing the behaviour of one or more agents and their management of resources.

- Where do RE methods fit into RE processes?
  - each method is appropriate for some particular types of problem domain
    - often not well-defined where they fit
  - methods vary in their coverage (of RE activities) and focus; e.g.,
    - Coverage: elicitation, modelling, analysis, etc.
    - Focus: goals, behaviour, viewpoints, etc.

What vs. How

- Traditionally, Requirements should specify 'what' without specifying 'how'
  - But this is not always easy to distinguish:
    - What does a car do?
    - What does a web browser do?
    - What does an operating system do?
  - The 'how' at one level of abstraction forms the 'what' for the next level
- Jackson's work provides a clearer distinction
  - 'What' refers to a system's purpose
    - it is external to the system
    - it is a property of the *application domain*
  - 'How' refers to a system's structure and behavior
    - it is internal to the system
    - it is a property of the *machine domain*



Source: Adapted from Jackson, 1995, p207

What are requirement about?

- Some distinctions:
  - Domain Properties are things in the application domain that are true whether or not we ever build the proposed system
  - Requirements are things in the application domain that we wish to be made true by delivering the proposed system
  - A specification is a description of the behaviours the program must have in order to meet the requirements
- Two verification criteria:
  - The Program running on a particular Computer satisfies the Specification
  - The Specification, in the context of the given Domain properties, satisfies the Requirements
- Two validation criteria:
  - Did we discover (and understand) all the important Requirements?
  - Did we discover (and understand) all the relevant Domain properties?

Source: Adapted from Jackson, 1995, p170-171

Validation Example

- Requirement R:
  - "Reverse thrust shall only be enabled when the aircraft is moving on the runway"
- Domain Properties D:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- Specification S:
  - Reverse thrust enabled if and only if wheel pulses on
- S + D imply R
  - But what if the domain model is wrong?

Source: Adapted from Jackson, 1995, p172

Another Example

- Requirement R:
  - "The database shall only be accessible by authorized personnel"
- Domain Properties D:
  - Authorized personnel have passwords
  - Passwords are never shared with non-authorized personnel
- Specification S:
  - Access to the database shall only be granted after the user types an authorized password
- S + D imply R
  - But what if the domain assumptions are wrong?

Source: Adapted from Jackson, 1995, p172

RE is all about Description

→ A designation
- singles out a phenomena of interest
  - tells you how to recognize it and gives it a name
- A designation is always informal,
  - it maps from the fuzzy phenomena to formal language

→ A definition
- gives a formal definition of a term that may be used in other descriptions
  - Note: definitions can be more or less useful, but never right or wrong.

→ A refutable description
- states some property of a domain that could in principle be refuted
  - Might not be practical to refute it, but refutation should be conceivable
- Refutability depends on an appeal to the designated phenomena of the domain being described

→ A rough sketch
- is a tentative description that is being developed

Source: Adapted from Jackson, 1995, p58-59

Examples

→ Designations:
- Parent(x,p) denotes that p is the genetic parent of x
- Female(x) denotes that x is biologically female
- …

→ Definitions:
- mother(x,m) ≙ Parent(x,m) and Female(m)
- sister(x,y) ≙ Female(x) and mother(x,m) and mother(y,m) and father(x,f) and father (y,f)

→ Refutable Description:
- For all m and x, Parent(x, p) implies not(Parent(m, p))

→ A rough sketch
- "Everyone's related somehow"

Source: Adapted from Jackson, 1995, p58-59

Requirements are optative

→ Traditionally, requirements contain the word 'shall'
- (and contractually, 'will' means it's optional!)
- The distinction in English is subtle:
  - "I shall drown. No one will save me"
  - "I will drown. No one shall save me"

→ Mood (of a verb):
- Indicative: asserts a fact ("you sing")
- Interrogative: asks a question ("are you singing")
- Imperative: conveys a command ("Sing!")
- Subjunctive: states a possibility ("I might sing")
- Optative: expresses a wish ("may you sing")

→ For requirements engineering:
- use the indicative mood for domain properties
- use the optative mood for requirements
- Never mix moods in the same description.
- Anyway, mood changes as development progresses! :-)

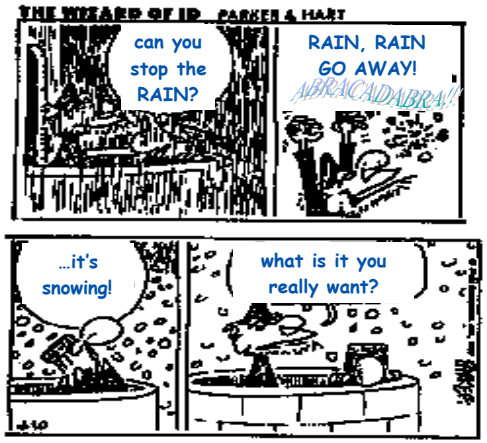
Source: Adapted from Jackson, 1995, p125-127

Phenomena

→ A little Philosophy:
- Phenomenology
  - the study of the things that appear to exist when you observe the world
- Ontology
  - the study of what really does exist (independently from any observer)
- Epistemology
  - the study of what people are capable of knowing (or what they believe)
- Weltanschauung
  - a world view that defines the set of phenomena that an observer is willing (likely) to observe ('viewpoint')

→ Each method has its own Weltanschauung
- Examples:
  - OO sees the world as objects with internal state that respond to stimuli
  - SA sees the world as processes that transform data
  - Natural language also defines a viewpoint
- Each method restricts the set of phenomena you can describe
  - …and therefore what you can model
- Choose a method that emphasizes the appropriate kinds of phenomena

Source: Adapted from Jackson, 1995, p143, and Blum 1996, chapter 2



Systems Theory

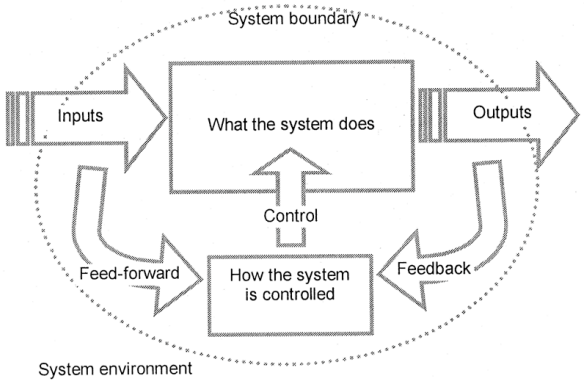
What is a system?

→ Definition of a System:
- Some part of reality that can be observed to interact with its environment
  - Separated from its environment by a boundary
  - A system receives inputs from the environment & send outputs to the environment
  - A system usually have subsystems
  - Systems that endure have a control mechanism
  - Systems have interesting emergent properties
- Examples:
  - cars, cities, houseplants, rocks, spacecraft, buildings, weather,...
  - operating systems, DBMS, the internet, an organization
- Non-examples (there aren't many!):
  - numbers, truth values, letters.
- *A closed system* doesn't interact with its environment (there aren't many!)

→ Systems might have no physical existence
- Only manifestations are symbolic/analogical representations of the system
- Such systems are social constructs: they exist because we agree on ways to observe them

Source: Adapted from Wieringa, 1996, p10

Conceptual picture of a system



Types of System

- Natural Systems
  - E.g. ecosystems, weather, water cycle, the human body, bee colony, ...
- Abstract Systems
  - E.g. set of mathematical equations, computer programs, etc
- Designed Systems
  - E.g. cars, planes, buildings, interstates, telephones, the internet, ...
- Human Activity Systems
  - E.g. Organizations, markets, clubs, …
- Information Systems (exist to support a HAS)
  - E.g. MIS, transaction processing, real-time control systems,…

Source: Adapted from Carter et. al., 1988, p12

Hard vs. Soft Systems

Hard Systems:

- The system is
  - precise,
  - well-defined
  - quantifiable
- No disagreement about:
  - Where the boundary is
  - What the interfaces are
  - The internal structure
  - Control mechanisms
  - The purpose (??)
- Examples
  - ?

Soft Systems:

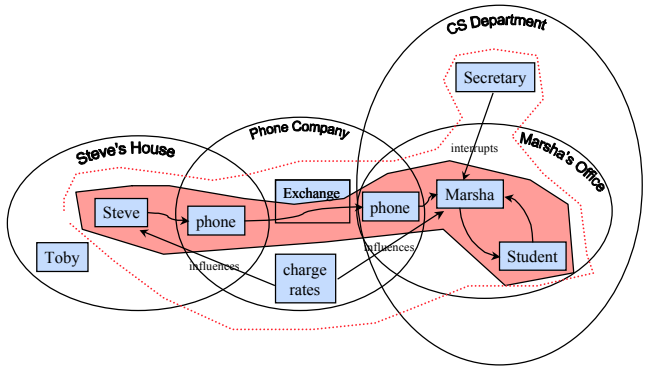
- The system…
  - …is hard to define precisely
  - …is an abstract idea
  - …depends on your perspective
- Not easy to get agreement
  - The system doesn't "really" exist
  - Calling something a system helps us to understand it
  - Identifying the boundaries, interfaces, controls, helps us to predict behaviour
  - The "system" is a theory of how some part of the world operates
- Examples:
  - All human activity systems
  - (what else?)

System Boundary

- System Environment:
  - the part of the world with which the system can interact
    - every system has an environment
    - the environment is itself a system
  - Distinction between system and environment depends on your viewpoint
- Choosing the boundary
  - Choice should be made to maximize modularity
  - Examples:
    - Telephone system - include: switches, phone lines, handsets, users, accounts?
    - Desktop computer - do you include the peripherals?
    - Flight control system - do you include the ground control?
  - Tips:
    - Exclude things that have no functional effect on the system
    - Exclude things that influence the system but which cannot be influenced or controlled by the system
    - Include things that can be strongly influenced or controlled by the system
    - Balance between totally open and totally closed systems

Source: Adapted from Wieringa, 1996, p11-12

Example System Boundary



Source: Adapted from Carter et. al., 1988, p6

Achieving Modularity

- Guidelines:
  - does the system have an underlying idea that can be described in one or two sentences?
  - Interaction among system components should be greater than interaction between the system and it's environment
    - Changes within a system should cause minimal changes outside
    - More 'energy' is required to transfer something across the system boundary than within the system boundary
  - The system boundary should 'divide nature at its joints'
- Choose the boundary that:
  - increases regularities in the behaviour of the system
  - simplifies the system behavior

Source: Adapted from Wieringa, 1996, p12

Control

- Control holds a system together
  - A system with no control won't endure
- A system can be characterized by the kind of control present
  - Self-maintaining causal network
    - a self-enhancing process: e.g. growth of the internet
    - a self-confirming process: e.g. visibility of a footpath
    - a self-limiting process: e.g. pricing of commodities
  - Purposive Control
    - System has a recognizable purpose or goal
    - control of sub-systems is directed towards achieving this goal
    - "purpose without choice"
  - Purposeful Control
    - special arrangements exist for decision making and control
    - Free choice among competing alternatives
    - "purpose with choice"

Source: Adapted from Carter et. al., 1988, p16

System Structure

- Subsystems…
  - A system is an organised collection of subsystems acting as a whole
    - subsytems are systems too!
  - Subsystem boundaries should be chosen so that subsystems are modular
- An *Aspect* of a system
  - is a restricted subset of the interactions between its subsystems
    - E.g. for a car: all interactions to do with safety
    - note fluidity between safety as an aspect, and safety as a subsystem
- Visibility
  - Interactions between subsystems only are *internal* to the system
  - Interactions between subsystems and the environment are *external*
  - Engineers usually try to hide internal interactions
    - For social systems, the internal interactions can be hidden too.
- Observability
  - the state space is defined in terms of the observable behavior
  - the perspective of the observer determines which states are observable

Source: Adapted from Wieringa, 1996, p13

System State

- State
  - a system will have memory of its past interactions, i.e. 'state'
  - the state space is the collection of all possible states
- Discrete vs continuous
  - a discrete system:
    - the states can be represented using natural numbers
  - a continuous system:
    - state can only be represented using real numbers
  - a hybrid system:
    - some aspects of state can be represented using natural numbers
- Observability
  - the state space is defined in terms of the observable behavior
  - the perspective of the observer determines which states are observable

Source: Adapted from Wieringa, 1996, p16-17

System Properties

- **A system property**
  - is an aspect of system behavior
    - often referred to as 'attributes' or 'quality attributes'
    - in software engineering, also known as the "ilities"
- **Specifying properties:**
  - A property is specified behaviorally if an experiment has been specified that will tell us unambiguously whether the system has the property
    - A property is specified non-behaviorally if no such experiment has been identified
  - Compare with: functional vs. non-functional requirements
  - Testing for non-behavioral properties requires a subjective (consensual) decision
- **Proxies**
  - Sometimes it is hard to specify a desired property behaviorally
    - can use a different property to indicate the presence of the desired property
  - E.g. 'easy to learn', 'easy to use' as proxies for 'user friendly'

Source: Adapted from Wieringa, 1996, p20-21

Systems Thinking