Lecture 2: Context for RE

Last Week:
INTRO
Syllabus
Course Goals
Definitions

This Week:
Context for RE
What is Engineering?
Types of engineering project
RE in the engineering lifecycle
Systems Thinking

Next Week:
Project Starting points:
(Stakeholders, Boundaries, Goals, Scenarios, Risks)

What is engineering?

“Engineering is the development of cost-effective solutions to practical problems, through the application of scientific knowledge”

“...Cost-effective...”
% Consideration of design trade-offs, esp. resource usage
% Minimize negative impacts (e.g. environmental and social cost)

“... Solutions ...”
% Emphasis on building devices

“... Practical problems ...”
% solving problems that matter to people
% improving human life in general through technological advance

“... Application of scientific knowledge ...”
% Systematic application of analytical techniques

Is software different?

→ Normal design:
% Old problems, whose solutions are well known
  ➢ Engineering codifies standard solutions
  ➢ Engineer selects appropriate methods and technologies
% Design focuses on well understood devices
  ➢ Devices can be studied independent of context
  ➢ Differences between the mathematical model and the reality are minimal

→ Radical design:
% Never been done, or past solutions have failed
  ➢ Often involves a very complex problem
% Bring together complex assemblies of devices into new systems
  ➢ Such systems are not amenable to reductionist theories
  ➢ Such systems are often soft: no objective criteria for describing the system

→ Examples:
  ➢ Most of Computer Engineering involves normal design
  ➢ All of Systems Engineering involves radical design (by definition)
  ➢ Much of Software Engineering involves radical design (soft systems)

→ Software is different!
% software is invisible, intangible, abstract
  ➢ Software alone is useless - its purpose is to configure some hardware to do something
% there are no physical laws underlying software behaviour
% there are no physical constraints on software complexity
% software never wears out
  ➢ ...traditional reliability measures don't apply
% software can be replicated perfectly
  ➢ ...no manufacturing variability

→ Software Myths:
% Myth: Cost of software is lower than cost of physical devices
% Myth: Software is easy to change
% Myth: Computers are more reliable than physical devices
% Myth: Software can be formally proved to be correct
% Myth: Software reuse increases safety and reliability
% Myth: Computers reduce risk over mechanical systems
Professional Responsibility

→ ACM/IEEE code of ethics:
% PUBLIC - act consistently with the public interest.
% CLIENT AND EMPLOYER - act in a manner that is in the best interests of your client and employer, consistent with the public interest.
% PRODUCT - ensure that your products and related modifications meet the highest professional standards possible.
% JUDGEMENT - maintain integrity and independence in your professional judgment.
% MANAGEMENT - subscribe to and promote an ethical approach to the management of software development and maintenance.
% PROFESSION - advance the integrity and reputation of the profession consistent with the public interest.
% COLLEAGUES - be fair to and supportive of your colleagues.
% SELF - participate in lifelong learning and promote an ethical approach to the practice of the profession.

→ Of particular relevance in RE:
% Competence - never misrepresent your level of competence
% Confidentiality - respect confidentiality of all stakeholders
% Intellectual property rights - respect protections on ideas and designs
% Data Protection - be aware of relevant laws on handling personal data

Project Management

→ A manager can control 4 things:
% Resources (can get more dollars, facilities, personnel)
% Time (can increase schedule, delay milestones, etc.)
% Product (can reduce functionality - e.g. scrub requirements)
% Risk (can decide which risks are acceptable)

→ To do this, a manager needs to keep track of:
% Effort - How much effort will be needed? How much has been expended?
% Time - What is the expected schedule? How far are we deviating from it?
% Size - How big is the planned system? How much have we built?
% Defects - How many errors are we making? How many are we detecting? And how do these errors impact quality?

→ Initially, a manager needs good estimates
% ...and these can only come from a thorough analysis of the problem.
You cannot control that which you cannot measure!

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 4GL.
  Examples: Payroll and personnel, Financial transactions, Customer relations database, ...

→ Control Systems
% software that drives some sort of a hardware process
  Examples: flight control, industrial plant, an elevator system, credit card reader.

→ Generic Services
% systems that provide some services for other systems
  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.
Project Context

→ Existing System
  % There is nearly always an existing system
  ➢ May just be a set of ad hoc workarounds for the problem
  % Studying it is important:
  ➢ If we want to avoid the weaknesses of the old system...
  ➢ ...while preserving what the stakeholders like about it

→ Pre-Existing Components
  % Benefits:
  ➢ Can dramatically reduce development cost
  ➢ Easier to decompose the problem if some subproblems are already solved
  % Tension:
  ➢ Solving the real problem vs. solving a known problem (with ready solution)

→ Product Families
  % Vertical families: e.g. 'basic', 'deluxe' and 'pro' versions of a system
  % Horizontal families: similar systems used in related domains
  ➢ Need to define a common architecture that supports anticipated variability

Lifecycle of an Engineering Project

→ Lifecycle models
  % Useful for comparing projects in general terms
  % Not enough detail for project planning

→ Examples:
  % Sequential models: Waterfall, V model
  % Rapid Prototyping
  % Phased Models: Incremental, Evolutionary
  % Iterative Models: Spiral
  % Agile Models: eXtreme Programming

→ Comparison: Process Models
  % Used for capturing and improving the development process

Waterfall Model

→ View of development:
  % a process of stepwise refinement
  % largely a high level management view
→ Problems:
  % 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.

V-Model

Source: Adapted from Herbsleb, 1997, p7 & Loucopoulos & Karakostas, 1995, p29
Prototyping lifecycle

- Understanding 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

The Spiral Model

Plan
- Determine goals, alternatives, constraints
- Evaluate alternatives and risks

Develop and test
- Implementation plan
- Integration and test plan
- Development plan

Source: Adapted from Dorfman, 1997, pp9

Phased Lifecycle Models

Incremental development
- Each release adds more functionality

Evolutionary development
- Each version incorporates new requirements

Agile Models

Basic Philosophy
- Reduce communication barriers
  - Programmer interacts with customer
- Reduce document-heavy approach
  - Documentation is expensive and of limited use
- Have faith in the people
  - Don't need fancy process models to tell them what to do
- Respond to the customer
  - Rather than focusing on the contract

Weaknesses
- Relies on programmer's memory
  - Code can be hard to maintain
- Relies on oral communication
  - Mis-interpretation possible
- Assumes single customer representative
  - Multiple viewpoints not possible
- Only short term planning
  - No longer term vision

E.g. Extreme Programming
- Instead of a requirements spec, use:
  - User story cards
  - On-site customer representative
  - Pair Programming
  - Small releases
    - E.g. every three weeks
  - Planning game
    - Select and estimate user story cards at the beginning of each release
- Write test cases before code
- The program code is the design doc
  - Can also use CRC cards (Class-Responsibility-Collaboration)
- Continuous Integration
  - Integrate and test several times a day
**Extreme Programming**

- Collect User stories
- Planning game
- Write test cases
- code
- integrate
- test

Each cycle: approx 2 weeks

**Is there a “Requirements Lifecycle”**

<table>
<thead>
<tr>
<th>Specification</th>
<th>Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete</td>
<td></td>
</tr>
<tr>
<td>fair</td>
<td></td>
</tr>
<tr>
<td>vague</td>
<td></td>
</tr>
<tr>
<td>personal view</td>
<td>common view</td>
</tr>
<tr>
<td>informal</td>
<td>semi-formal</td>
</tr>
</tbody>
</table>

**Inquiry Cycle**

- **Prior Knowledge** (e.g. customer feedback)
- Initial hypothesis
- **Observe** (what is wrong with the current system?)
  - Look for anomalies - what can't the current theory explain?
- **Intervene** (replace the old system)
  - Carry out the experiments
- **Model** (describe/explain the observed problems)
  - Create/Refine a better theory
- **Design** (invent a better system)
  - Design experiments to test the new theory

Note similarity with Process of scientific investigation:
Requirements models are theories about the world; Designs are tests of those theories.

**The Wizard of Oz:**

- can you stop the RAIN?
- RAIN, RAIN GO AWAY!
- ...it’s snowing!
- what is it you really want?
The story so far:

→ What is engineering?
  % Not that different from science
  % Greater awareness of professional responsibility
  ➢ because of immediate scope for harm to the public
  % Systems and Software Engineering involve radical design

→ Engineering Projects
  % You cannot control that which you cannot measure
  ➢ ...and many important measures are derived from initial problem analysis
  % Constraints:
    ➢ Is there a customer?
    ➢ Existing system / existing components / existing product family

→ Project Lifecycles
  % Useful for comparing projects in general terms
  % Represent different philosophies in software development
  % Requirements evolve through their own lifecycles too

General Systems Theory

→ How scientists understand the world:
  % Reductionism - break a phenomena down into its constituent parts
    ➢ E.g. reduce to a set of equations governing interactions
  % Statistics - measure average behaviour of a very large number of instances
    ➢ E.g. gas pressure results from averaging random movements of zillions of atoms
    ➢ Error tends to zero when the number of instances gets this large

→ But sometimes neither of these work:
  % Systems that are too interconnected to be broken into parts
  % Behaviour that is not random enough for statistical analysis

→ General systems theory
  % Originally developed for biological systems:
    ➢ E.g. to understand the human body, and the phenomena of 'life'
  % Basic ideas:
    ➢ Treat inter-related phenomena as a system
    ➢ Study the relationships between the pieces and the system as a whole
    ➢ Don’t worry if we don’t fully understand each piece

Role of the Observer

→ Achieving objectivity in scientific inquiry
  1. Eliminate the observer
    ➢ E.g. ways of measuring that have no variability across observers
  2. Distinguish between scientific reasoning and value-based judgement
    ➢ Science is (supposed to be) value-free
    ➢ (but how do scientists choose which theories to investigate?)

→ For complex systems, this is not possible
  % Cannot fully eliminate the observer
    ➢ E.g. Probe effect - measuring something often changes it
    ➢ E.g. Hawthorne effect - people react to being studied
  % Our observations biased by past experience
    ➢ We look for familiar patterns to make sense of complex phenomena
    ➢ E.g. try describing someone’s accent

→ Achieving objectivity in systems thinking
  % Study the relationship between observer and observations
  % Look for observations that make sense from many perspectives
Relativism

→ Truth is relative to many things
  % The meanings of the words we use
    ➢ E.g. law of gravity depends on correct understanding of “mass”, “distance”, “force” etc
  % The assumptions we make about context
    ➢ E.g. law of gravity not applicable at subatomic level, or near the speed of light
    ➢ E.g. Which is the step function:

Relativism is everywhere

→ Truth often depends on the observer
  % “Emergent properties of a system are not predictable from studying the parts”
    ➢ Whose ability to predict are we talking about?
  % “Purpose of a system is a property of the relationship between system & environment”
    ➢ What is the purpose of: General Motors? A University? A birthday party?

The agricultural revolution
Transistor switching

→ Weltanshaungen (≈ “worldviews”)
  % Our Weltanshaungen permeate everything
    ➢ The set of categories we use for understanding the world
    ➢ The language we develop for describing what we observe

→ Ethno-centrism (or ego-centrism)
  % The tendency to assume one’s own category system is superior
    ➢ E.g. “In the land of the blind, the one-eyed man is king”
    ➢ But what use would visually-oriented descriptions be in this land?

The principle of complementarity

→ Raw observation is too detailed
  % We systematically ignore many details
    ➢ E.g. the idea of a ‘state’ is an abstraction
  % All our descriptions (of the world) are partial, filtered by:
    ➢ Our perceptual limitations
    ➢ Our cognitive ability
    ➢ Our personal values and experience

→ Complementarity:
  % Two observers’ descriptions of system may be:
    ➢ Redundant - if one observer’s description can be reduced to the other
    ➢ Equivalent - if redundant both ways
    ➢ Independent - if there is no overlap at all in their descriptions
    ➢ Complementary - if none of the above hold
  % Any two partial descriptions (of the same system) are likely to be complementary
  % Complementarity should disappear if we can remove the partiality
    ➢ E.g. ask the observers for increasingly detailed observations
  % But this is not always possible/feasible

Definition of a system

→ Ackoff’s definition:
  % “A system is a set of two or more elements that satisfies the following conditions:
    ➢ The behaviour of each element has an effect on the behaviour of the whole
    ➢ The behaviour of the elements and their effect on the whole are interdependent
    ➢ However subgroups of elements are formed, each has an effect on the whole and none has an independent effect on it”

→ Other views:
  % Weinberg: “A system is a collection of parts, none of which can be changed on its own”
    ➢ ...because the parts of the system are so interconnected
  % Wierringa: “A system is any actual or possible part of reality that, if it exists, can be observed”
    ➢ ...suggests the importance of an observer
  % Weinberg: “A system is a way of looking at the world”
    ➢ Systems don’t really exist!
    ➢ Just a convenient way of describing things (like ‘sets’)
Elements of a system

- **Boundary**
  - Separates a system from its environment
  - Often not sharply defined
  - Also known as an “interface”

- **Environment**
  - Part of the world with which the system can interact
  - System and environment are inter-related

- **Observable Interactions**
  - How the system interacts with its environment
  - E.g., inputs and outputs

- **Subsystems**
  - Can decompose a system into parts
  - Each part is also a system
  - For each subsystem, the remainder of the system is its environment
  - Subsystems are inter-dependent

- **Control Mechanism**
  - How the behaviour of the system is regulated to allow it to endure
  - Often a natural mechanism

- **Emergent Properties**
  - Properties that hold of a system, but not of any of the parts
  - Properties that cannot be predicted from studying the parts

**Conceptual Picture of a System**

*System Boundary*

- Inputs
- How the system's properties evolve over time
- Outputs
- Feedback

**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:
  - A car (?)

**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

**Types of System**

- **Natural Systems**
  - E.g., ecosystems, weather, water cycle, the human body, bee colony, ...
  - Usually perceived as hard systems

- **Abstract Systems**
  - E.g., set of mathematical equations, computer programs, ...
  - Interesting property: system and description are the same thing

- **Symbol Systems**
  - E.g., languages, sets of icons, street signs, ...
  - Soft because meanings change

- **Designed Systems**
  - E.g., cars, planes, buildings, freeways, telephones, the internet, ...

- **Human Activity Systems**
  - E.g., businesses, organizations, markets, clubs, ...
  - E.g., any designed system when we also include its context of use
    - Similarly for abstract and symbol systems!

- **Information Systems**
  - Special case of designed systems
    - Part of the design includes the representation of the current state of some human activity system
    - E.g., MIS, banking systems, databases, ...

- **Control systems**
  - Special case of designed systems
    - Designed to control some other system (usually another designed system)
    - E.g., thermostats, autopilots, ...
### Information Systems

- **Needs information about**
- **Maintains information about**
- **Uses**
- **Builds**

- **Subject System**
- **Usage System**
- **Information System**
- **Development System**

### Control Systems

- **Needs to ensure safe control of**
- **Tracks and controls the state of**

- **Subject System**
- **Usage System**
- **Information System**
- **Development System**

---

### Software-Intensive Systems

- **Software-Intensive Human Activity System**
- **Information System**
- **Computer-Based Information System**
- **Subject System**
- **Hardware System**
- **Software System**
- **Utility Language System**

### Open and Living Systems

**Openness**
- The degree to which a system can be distinguished from its environment
- A closed system has no environment
  - If we describe a system as closed, we ignore its environment
  - E.g., an egg can be described as a closed system
- A fully open system merges with its environment

**Living systems**
- Special kind of open system that can preserve its identity and reproduce
  - Also known as “neg-entropy” systems
  - E.g., biological systems
  - Reproduction according to DNA instructions
  - E.g., Social systems
  - Rules of social interaction act as a kind of DNA
**Purposefulness**

- **Types of behaviours:**
  - Reaction to a stimulus in the environment
  - Response to a stimulus in the environment
  - Autonomous act: A system event for which a stimulus is not necessary

- **Systems can be:**
  - State-maintaining
  - Goal-directed
  - Purposive

**System has multiple goals, can choose how to pursue them, but no choice over the goals themselves**

- **Purposeful:**
  - System has multiple goals, and can choose to change its goals
  - E.g. people, governments, businesses, animals

**Scoping a system**

- **Choosing the boundary**
  - Distinction between system and environment depends on your viewpoint
  - Choice should be made to maximize modularity

- **Examples:**
  - Telephone system - include: switches, phone lines, handsets, users, accounts?
  - Desktop computer - do you include the peripherals?

- **Tips:**
  - Exclude things that have no functional effect on the system
  - Include 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
  - 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

- **System boundary should 'divide nature at its joints'**
  - Choose the boundary that:
    - increases regularities in the behaviour of the system
    - simplifies the system behavior

**Example Scoping Problem**

**Layers of systems**

<table>
<thead>
<tr>
<th>appropriate for:</th>
<th>Subsystems</th>
<th>System</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis of repair problems</td>
<td>Wires, connectors, receivers</td>
<td>Subscriber's household phone system</td>
<td>Telephone calls.</td>
</tr>
<tr>
<td>Analysis of individual phone calls</td>
<td>Subscribers' phone systems</td>
<td>Telephone calls</td>
<td>Regional phone network</td>
</tr>
<tr>
<td>Analysis of regional sales strategy</td>
<td>Telephone calls</td>
<td>Regional phone network</td>
<td>National telephone market and trends</td>
</tr>
<tr>
<td>Analysis of phone company's long term planning</td>
<td>Regional phone networks</td>
<td>National telephone market and trends</td>
<td>Global communication systems</td>
</tr>
</tbody>
</table>
Describing System Behaviour

→ 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

Summary: Systems Thinking

Source: Adapted from Wieringa, 1996, p16-17