Lecture 2: Context for RE

Last Week:
- INTRO
- Syllabus
- Course Goals
- Definitions

Next Week:
- Project Starting points: {Stakeholders, Boundaries, Goals, Scenarios, Risks}

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

What is Engineering?

- **Definition of Engineering:**
  - Engineering is the development of cost-effective solutions to practical problems, through the application of scientific knowledge
  - "...cost-effective..." - involves trade-offs, especially with resource usage
  - "...solutions..." - engineering is creative and interventionist
  - "...practical problems..." - the problems must matter to people
  - "...scientific knowledge..." - uses analytical techniques based on applied science

- **Normal or Radical design?**
  - 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
  - Radical design: never been done, or past solutions have failed
    - Often the challenge is to deal with a very complex problem
    - Bring together complex assemblies of devices into new systems
    - Systems Engineering is always radical design (by definition!)

Is software different?

- 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 key variables:
- 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)

Approach (applies to any management)
- Understand the goals and objectives
  - quantify them where possible
- Understand the constraints
  - if there is uncertainty, use probability estimates
- Plan to meet the objectives within the constraints
- Monitor and adjust the plan
- Preserve a calm, productive, positive work environment

Note:
- You cannot control what you cannot measure!

Where Projects Come From

- Initiation of the project
  - Problem-driven
    - A problem has arisen that demands a response
    - e.g. existing system is "broken"
  - Change-driven
    - Changes in the business or its environment
    - existing system becoming less useful
  - Opportunity-driven
    - New technology opens up new possibilities:
    - New markets open up:
    - etc
  - Legacy-driven
    - Project created because of prior commitment
    - e.g. earlier work left unfinished

Source of Requirements:
- Customer-specific
  - Specific customer with a specific problem
  - The customer is the ultimate authority
- Market-based
  - System designed to be sold widely
  - Marketing team acts as proxy for customers & users
- Product must generate customers
- Socially-useful
  - System is intended as a general benefit to society
  - No (paying) customer
  - E.g. some open source / free software; software created in scientific research
- Hybrid
  - developed for a specific customer, but want to market the software eventually

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 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.

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.
**Prototyping lifecycle**

- **requirements**
- **design**
- **build**
- **test**
- **prototype**

**Requirements in the Prototyping lifecycle**

- **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**

- **Incremental development**
  - (each release adds more functionality)
  - **version 1**
  - **version 2**
  - **version 3**
  - **version 4**

- **Evolutionary development**
  - (each version incorporates new requirements)

**The Spiral Model**

- **Determine goals, alternatives, constraints**
- **Evaluate alternatives and risks**
- **Plan**
- **Develop and test**

**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
V-Model

- System requirements
- Software requirements
- Preliminary design
- Detailed design
- Component test
- Code and debug
- Unit test
- Software integration
- System integration
- Acceptance test

"Analyse and design"

"Test and integrate"

Agile Models

Basic Philosophy
- Reduce communication barriers
- Reduce document-heavy approach
- Have faith in the people
- Respond to the customer

Weakeness
- Relies on programmer's memory
- Assumessingle customer representative
- Assumes single customer representative
- Only short term planning

E.g. Extreme Programming
- Instead of a requirements spec, use:
  - User story cards
  - On-site customer representative
  - Pair Programming
  - Small releases
  - Planning game

- Write test cases before code
- The program code is the design doc
- Can also use CRC cards (Class-Responsibility-Collaboration)
- Continuous Integration

Inquiry Cycle

Prior Knowledge (e.g. customer feedback)

Observe (what is wrong with the current system?)

Intervene (replace the old system)

Model (describe/explain the observed problems)

Design (invent a better system)

Note similarity with Process of scientific Investigation:
- Requirements models are theories about the world
- Designs are tests of those theories

...it's snowing!

Look for anomalies - what can't the current theory explain?

Create/refine a better theory

Can you stop the RAIN?
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 & sends 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, p18

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
  - 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

Example System Boundary

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 its 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
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"

System Structure

- Subsystems...
  - A system is an organised collection of subsystems acting as a whole
    - subsystems 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

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
- For modelling purposes:
  - Can approximate a continuous system with a discrete model
  - All models are approximations anyway!
  - But make sure the inaccuracies don’t matter...

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’
Systems Thinking

A system that helps to understand the real-world situation

Thinks about

Makes comparisons

A real-world situation or problem

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

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