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, streetsigns, ...
  - 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, ...
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

Which systems are soft?

• **Generic software components**
  – E.g. Core operating system functions, network services, middleware, …
  – Functionality relatively stable, determined by technical interfaces
  – But note that these systems still affect human activity
    • E.g. concepts of a ‘file’, a ‘URL’, etc.

• **Control Systems**
  – E.g. aircraft flight control, industrial process control, …
  – Most requirements determined by the physical processes to be controlled
  – But note that operator interaction is usually crucial
    • E.g. accidents caused when the system doesn’t behave as the operator expected

• **Information Systems**
  – E.g. office automation, groupware, web services, business support,…
  – These systems cannot be decoupled from the activities they support
  – Design of the software entails design of the human activity
    • The software and the human activities co-evolve
Software-Intensive Systems

• **Software (on its own) is useless**
  - Software is an abstract description of a set of computations
  - Software only becomes useful when run on some hardware
    • we sometimes take the hardware for granted
  - **Software + Hardware = “Computer System”**

• **A Computer System (on its own) is useless**
  - Only useful in the context of some human activity that it can support
    • we sometimes take the human context for granted
  - A new computer system will change human activities in significant ways
  - **Software + Hardware + Human Activities = “Software-Intensive System”**

• ‘**Software**’ makes many things possible
  - It is complex and adaptable
  - It can be rapidly changed on-the-fly
  - It turns general-purpose hardware into a huge variety of useful machines

---

**Is software different?**

• **Software is different!**
  - software is invisible, intangible, abstract
    • its purpose is to configure some hardware to do something useful
  - there are no physical laws underlying software behaviour
  - there are no physical constraints on software complexity
  - 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
Control Systems

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

Usage System

- builds

Development System

- contracts

Control system

Information Systems

- Needs information about
- Maintains information about

Usage System

- contracts

Development System

- builds

Information system

Source: Adapted from Loucopoulos & Karakostas, 1995, p73
Lifecycle of 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

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 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)

Evolutionary development (each version incorporates new requirements)

The Spiral Model

Determine goals, alternatives, constraints

Evaluate alternatives and risks

Plan

Develop and test
Agile Models

Source: Adapted from Nawrocki et al., RE ’02

• 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

Extreme Programming

Instead of a requirements spec, use:
- User story cards
- On-site customer representative

Pair Programming
- Small releases
  - E.g. every two or 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

Release

Each cycle:
approx 2-3 weeks

code

write test cases

integrate

Planning game

Collect User stories

Write test cases

Each cycle: approx 2-3 weeks

Planning game

Each cycle: approx 2-3 weeks

Planning game

Each cycle: approx 2-3 weeks

Planning game
Cost of getting it wrong

• Cost of fixing errors
  – Typical development process:
    requirements analysis ⇒ software design ⇒ programming ⇒
    development testing ⇒ acceptance testing ⇒ operation
  – Errors cost more to fix the longer they are undetected
    • E.g. A requirements error found in testing costs 100 times more than
      a programming error found in testing

• Causes of project failure
  – Survey of US software projects by the Standish group:

<table>
<thead>
<tr>
<th></th>
<th>2004</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>16%</td>
<td>26%</td>
</tr>
<tr>
<td>Challenged</td>
<td>53%</td>
<td>46%</td>
</tr>
<tr>
<td>Cancelled</td>
<td>31%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Top 3 success factors:
1) User involvement
2) Executive management support
3) Clear statement of requirements

Top 3 factors leading to failure:
1) Lack of user input
2) Incomplete requirements & specs
3) Changing requirements & specs

Where are the challenges?
What does requirements engineer do?

• Explain the concepts, theories, and best practices associated with requirements engineering
• Elicit, negotiate and document software requirements
• Develop major requirements artefacts and use them during the software development projects
• Apply requirements validation techniques
• Manage software requirements, priorities, and trace them


• Works directly w/ stakeholders
• Elicits requirements
• Analyzes requirements
• Documents requirements
• May create functional specifications
What are requirements?

- **Domain Properties:**
  - things in the application domain that are true whether or not we ever build the proposed system

- **Requirements:**
  - things in the application domain that we wish to be made true by delivering the proposed system
    - Many of which will involve phenomena the machine has no access to

- **A Specification:**
  - is a description of the behaviours that the program must have in order to meet the requirements
    - Can only be written in terms of shared phenomena!

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
Definition of RE

Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive 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.

Not a phase or stage!

Communication is as important as the analysis

Quality means fitness-for-purpose. Cannot say anything about quality unless you understand the purpose

Designers need to know how and where the system will be used

Requirements are partly about what is needed...

...and partly about what is possible

Need to identify all the stakeholders - not just the customer and user

Some observations about RE

• RE is not necessarily a sequential process:
  – Don’t have to write the problem statement before the solution statement
    • (Re-)writing a problem statement can be useful at any stage of development
  – RE activities continue throughout the development process

• The problem statement will be imperfect
  – RE models are approximations of the world
    • will contain inaccuracies and inconsistencies
    • will omit some information.
    • analysis should reduce the risk that these will cause serious problems...

• Perfecting a specification may not be cost-effective
  – Requirements analysis has a cost
  – For different projects, the cost-benefit balance will be different

• Problem statement should never be treated as fixed
  – Change is inevitable, and therefore must be planned for
  – There should be a way of incorporating changes periodically
Things to take home

• Software-intensive systems
• Lifecycle of engineering projects
• Challenges of software projects
• Requirements engineering