MTAT.03.306

Lecture 1: Introduction

Prof. Steve Easterbrook, Requirements engineering course, University of Toronto
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, …
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
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
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
Control Systems

Subject system

Needs to ensure safe control of
contracts

Usage System

Uses

Tracks and controls the state of

Control system

Development System

builds
Information Systems

Source: Adapted from Loucopoulos & Karakostas, 1995, p73

- Subject System
  - Uses
  - Builds
  - Maintains information about

- Usage System
  - Needs information about
  - Contracts

- Development System
  - Information system

Diagram illustrates the interactions between the Subject System, Usage System, and Development System.
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>2008</th>
<th>2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>26%</td>
<td>16%</td>
</tr>
<tr>
<td>Challenged</td>
<td>46%</td>
<td>53%</td>
</tr>
<tr>
<td>Cancelled</td>
<td>28%</td>
<td>31%</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

https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
Where are the challenges?

Application Domain

Machine Domain

domain properties
requirements

specification

programs
computers
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.
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.

- 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.

Not a phase or stage!
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
Any questions