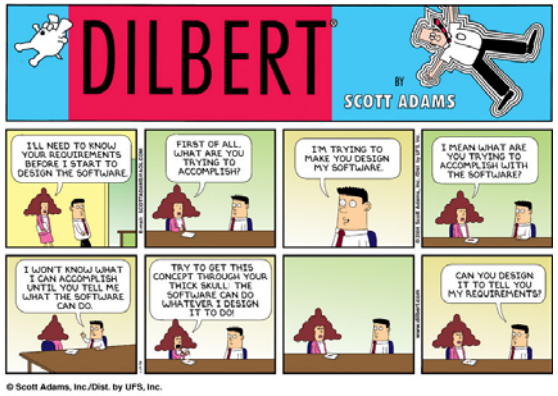


ECE450 – Software Engineering II

Today: **Introduction to Requirements Engineering**

adapted from Steve Easterbrook's material on Requirements Engineering

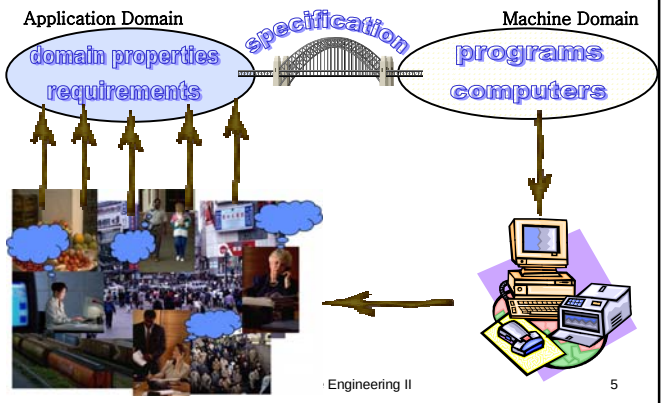
Building software *for a reason*

- 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

Quality = Fitness for purpose

- Software technology is everywhere
  - Affects nearly all aspects of our lives
  - But our experience of software technology is often frustrating/disappointing
- Software is **designed for a purpose**
  - If it doesn't work well then either:
    - …the designer didn't have an adequate understanding of the purpose
    - …or we are using the software for a purpose different from the intended one
  - ***Requirements analysis is about identifying this purpose***
    - Inadequate understanding of the purpose leads to poor quality software
- The purpose is found in human activities
  - E.g. Purpose of a banking system comes from the business activities of banks and the needs of their customers
  - The purpose is often complex:
    - Many different kinds of people and activities
    - Conflicting interests among them

Interaction loop



Complexity of purpose

- People and software are closely-coupled
  - Complex modes of interaction
  - Long duration of interaction
  - Mixed-initiative interaction
  - Socially-situated interaction
  - …software systems and human activity shape each other in complex ways
- The problems we'd like software to solve are "wicked"
  - No definitive formulation of the problem
  - No stopping rule (each solution leads to new insights)
  - Solutions are not right or wrong
  - No objective test of how good a solution is (subjective judgment needed)
  - Each problem is unique (no other problem is exactly like it)
  - Each problem can be treated as a symptom of another problem
  - Problems often have strong political, ethical or professional dimensions

Dealing with problem complexity

- Abstraction
  - Ignore detail to see the big picture
  - Treat objects as the same by ignoring certain differences
  - (beware: every abstraction involves choice over what is important)
- Decomposition
  - Partition a problem into independent pieces, to study separately
  - (beware: the parts are rarely independent really)
- Projection
  - Separate different concerns (views) and describe them separately
  - Different from decomposition as it does not partition the problem space
  - (beware: different views will be inconsistent most of the time)
- Modularization
  - Choose structures that are stable over time, to localize change
  - (beware: any structure will make some changes easier and others harder)

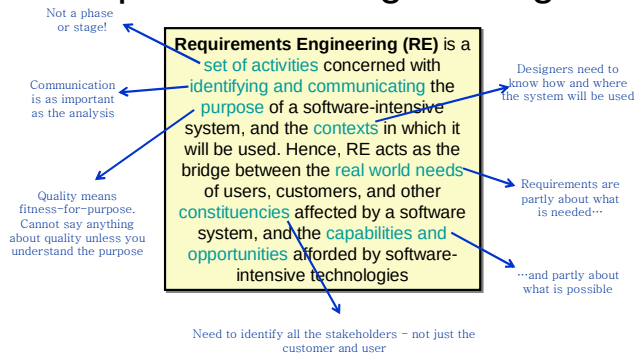
Designing for people

- What is the real goal of software design?
  - Creating new programs, components, algorithms, user interfaces,…?
  - Making human activities more effective, efficient, safe, enjoyable,…?
- How rational is the design process?
  - Hard systems view:
    - Software problems can be decomposed systematically
    - The requirements can be represented formally in a specification
    - This specification can be validated to ensure it is correct
    - A correct program is one that satisfies such a specification
  - Soft systems view:
    - Software development is embedded in a complex organizational context
    - There are multiple stakeholders with different values and goals
    - Software design is part of an ongoing learning process by the organization
    - Requirements can never be adequately captured in a specification
    - Participation of users and others throughout development is essential
  - Reconciliation:
    - Hard systems view okay if there is local consensus on the nature of the problem

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

Definition of Requirements Engineering



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:

	1994	1998
Successful	16%	26%
Challenged	53%	46%
Cancelled	31%	28%

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

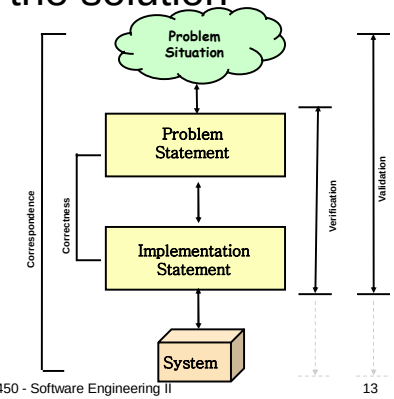
Top 3 factors leading to failure:
1) Lack of user input
2) Incomplete requirements & specs
3) Changing requirements & specs

What do Requirements Analysts do?

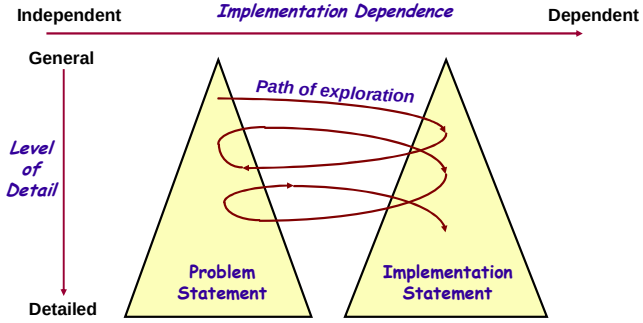
- Starting point
  - Some notion that there is a "problem" that needs solving
    - e.g. dissatisfaction with the current state of affairs
    - e.g. a new business opportunity
    - e.g. a potential saving of cost, time, resource usage, etc.
  - A Requirements Analyst is an agent of change
- The requirements analyst must:
  - identify the "problem"/"opportunity"
    - Which problem needs to be solved? (identify problem **Boundaries**)
    - Where is the problem? (understand the **Context**/Problem Domain)
    - Whose problem is it? (identify **Stakeholders**)
    - Why does it need solving? (identify the stakeholders' **Goals**)
    - How might a software system help? (collect some **Scenarios**)
    - When does it need solving? (identify Development **Constraints**)
    - What might prevent us solving it? (identify **Feasibility** and **Risk**)
  - and become an expert in the problem domain
    - although ignorance is important too -- "the smart ignoramus"

Separating the problem from the solution

- A separate problem description is useful:
  - Most obvious problem might not the right one to solve
  - Problem statement can be discussed with stakeholders
  - Problem statement can be used to evaluate design choices
  - Problem statement is a source of good test cases
- Still need to check:
  - Solution correctly solves the stated problem
  - Problem statement corresponds to the needs of the stakeholders



Intertwining of problems and solutions



Observations about Requirements Engineering

- It 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
  - Requirements Eng. activities continue throughout the development process
- The problem statement will be imperfect
  - Requirements 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

Revisiting Alice

- Nightmare scenario, yes, but the customer is not the only one at fault here!