Requirements Engineering: Modelling the User's Needs

Prof. Steve Easterbrook
Dept of Computer Science
University of Toronto
http://www.cs.toronto.edu/~sme

The “essential” software process

Real World

Problem Statement

Implementation Statement

System

Correspondence Correctness

Verification

Validation

Source: Adapted from Bhon, 1992, p32
see also: van Vliet p11
Design changes the world...

Discovering Requirements

- 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 Engineer is an agent of change

- The requirements engineer 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 intelligent ignoramus"
Difficulties of Elicitation

- Thin spread of domain knowledge
  - The knowledge might be distributed across many sources
  - It is rarely available in an explicit form (i.e., not written down)
  - There will be conflicts between knowledge from different sources
    - People have conflicting goals
    - People have different understandings of the problem

- Tacit knowledge (The "say-do" problem)
  - People find it hard to describe knowledge they regularly use
  - Descriptions may be inaccurate rationalizations of expert behaviour

- Limited Observability
  - The problem owners might be too busy coping with the existing system
  - Presence of an observer may change the problem
    - E.g., the Probe Effect and the Hawthorne Effect

- Bias
  - People may not be free to tell you what you need to know
    - Political climate & organisational factors matter
  - People may not want to tell you what you need to know
    - The outcome will affect them, so they may try to influence you (hidden agendas)

Requirements Growth

- Davis's model:
  - User needs evolve continuously
    - Imagine a graph showing growth of needs over time
    - May not be linear or continuous (hence no scale shown)
  - Traditional development always lags behind needs growth
    - First release implements only part of the original requirements
    - Functional enhancement adds new functionality
    - Eventually, further enhancement becomes too costly, and a replacement is planned
    - The replacement also only implements part of its requirements, and so on...
  - Conventional development
    - Identify requirements
    - First release
    - Enhancement phase
    - Frozen and replace
    - Longevity
    - Lateness
    - Inappropriateness
    - Adaptability
    - Time

Source: Adapted from Davis 1988, pp1453-1455
What does correctness mean?

Some distinctions:
- **Domain Properties**: are things in the application domain that are true whether or not we ever build the proposed system.
- **Requirements**: are things in the application domain that we wish to be made true by delivering the proposed system.
- A specification is a description of the behaviours the program must have in order to meet the requirements.

Two correctness criteria:
- The Program running on a particular Computer satisfies the Specification.
- The Specification, in the context of the given domain properties, satisfies the requirements.

Two validation criteria:
- We discovered all the important requirements.
- We discovered all the relevant domain properties.

---

Validation Example

**Requirement R:**
- "Reverse thrust shall only be enabled when the aircraft is moving on the runway."

**Domain Properties D:**
- Wheel pulses on if and only if wheels turning.
- Wheels turning if and only if moving on runway.

**Specification S:**
- Reverse thrust enabled if and only if wheel pulses on.

**S + D imply R**
- But what if the domain assumptions are wrong?
Another Example

- **Requirement R:**
  - “The database shall only be accessible by authorized personnel”

- **Domain Properties D:**
  - Authorized personnel have passwords
  - Passwords are never shared with non-authorized personnel

- **Specification S:**
  - Access to the database shall only be granted after the user types an authorized password

- **S + D imply R**
  - But what if the domain assumptions are wrong?

---

Setting the Boundaries

- **How will the software interact with the world?**

- **E.g. the four variable model:**
  - Fixes the input/output devices
  - Uses I/O data items as proxies for the monitored and controlled variables

---

Setting the Boundaries Diagram:

- **System**
  - **Input devices**
    - Monitored Variables
    - input data
  - **Output devices**
    - Controlled Variables
    - output data

- **Environment**
  - **Monitored Variables**
  - **Controlled Variables**

- **S** - Specification of software in terms of inputs & outputs
- **R** - Requirements: what control actions the system must take in which circumstances.
- **D** - Domain Properties that constrain how the environment can behave
Three different models??

- **R**: a model of the requirements
- **D**: a model of the environment
- **S**: a model of the software behaviour

Modeling...

- **Modeling can guide elicitation**
  - Does the modeling process help you figure out what questions to ask?
  - Does the modeling process help to surface hidden requirements?
    - i.e. does it help you ask the right questions?

- **Modeling can provide a measure of progress**
  - Does completeness of the model imply completeness of the elicitation?
    - i.e. if we've filled in all the pieces of the model, are we done?

- **Modeling can help to uncover problems**
  - Does inconsistency in the model reveal interesting things...
    - e.g. inconsistency could correspond to conflicting or infeasible requirements
    - e.g. inconsistency could mean confusion over terminology, scope, etc
    - e.g. inconsistency could reveal disagreements between stakeholders

- **Modeling can help us check our understanding**
  - Can we test that the model has the properties we expect?
  - Can we reason over the model to understand its consequences?
  - Can we animate the model to help us visualize/validate the requirements?
What models might we build?

(1) What kinds of information are we dealing with?
- What objects (classes of objects) are we dealing with?
- What are the relationships between those objects?
- What constraints are there on those relationships?

Example:

```
:car
VIN(vehicle Id Number) YearMade Mileage
0..*
owns
owner

:title
yearbought initialMileage PricePaid LicencePlate#

:person
Name Address DriversLicenceNumber PermittedVehicles
```

(2) How should objects interact?

ATM Model A

- Insert Card
- Prompt for PIN#
- Type PIN#
- Display Menu
- Request Cash
- Prompt for amount
- Enter amount
- Dispense Cash
- Confirm funds
- Withdraw funds
- Print Recent
- Display Menu
- End Transaction
- Return Card

ATM Model B

- Insert Card
- Prompt for PIN#
- Type PIN#
- Display Menu
- Request Cash
- Prompt for amount
- Enter amount
- Sufficient funds?
- Confirm funds
- Another Trans?
- Decline
- Return Card
- Dispense Cash
- Withdraw funds
- Print Receipt
What is the lifecycle of each object?

<table>
<thead>
<tr>
<th>:person</th>
<th>age</th>
<th>havebirthday()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**unborn**
- recordBirth()
- setDateOfBirth()

**child**
- havebirthday() [age < 18]
- havebirthday() [age = 18]

**adult**
- havebirthday() [age < 65]
- havebirthday() [age = 65]

**senior**
- havebirthday() [age < 65]

**deceased**
- recordDeath()
- setDateOfDeath()

A more detailed example

**unborn**
- registerBirth(setDateOfBirth())

**child**
- registerDeath() when [age > 17]

**adult**
- working age when [age > 65]

**senior**
- partnered
  - married when [addr = spouse.addr]
  - separated when [addr ≠ spouse.addr]

**single**
- unmarried
- widowed
- divorced

createRecord()
What other things matter?

Product operation
- usability
- integrity
- efficiency
- correctness
- reliability
- maintainability

Product revision
- testability
- flexibility
- reusability
- portability
- interoperability

Product transition
- operability
- training
- communicativeness
- I/O volume
- I/O rate
- Access control
- Access audit
- Storage efficiency
- execution efficiency
- traceability
- completeness
- accuracy
- error tolerance
- consistency
- simplicity
- conciseness
- instrumentation
- expandability
- generality
- Self-descriptiveness
- modularity
- machine independence
- s/w system independence
- comms. commonality
- data commonality

Summary