Requirements engineering

Principles and terminology

Domains and domain analysis

Examples

This material is taken from:

• M. Jackson, Software Requirements & Specifications
  Addison Wesley, 1995

• M. Jackson, P. Zave, Deriving Specifications from
  Requirements: An Example, Proceedings ICSE 1995

• C. Gunter et al. A Reference Model for Requirements and
The requirements phase

• Common to all engineering areas
  – e.g. bridge construction
    • in-site inspections
    • acquisition of data on geology of the area
    • expected traffic/load
    • ...

The problem and the solution

• Requirements deal with the problem space
• Design and implementation deal with the solution space
  – What versus how
Requirement

• Requirement for an artifact
  – Properties to be met by artifact in order to be considered successful
  – The effects that the machine has to exert in the problem domain by virtue of the software to be developed

• The engineering process
  – Bridging the gap between requirements and the available “raw materials” (languages, components, …)
The problem space

• Focus on the problem space
  – Understand the application domain
• e.g. automation of college registration office
  – admission
  – curricula & courses
  – taxes to pay, exemptions
  – ...
Requirements engineering

• The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed

• Requirements may be functional or non-functional
  – Functional requirements describe system services or functions
  – Non-functional requirements is a constraint on the system or on the development process
The requirements process

• Interaction required with
  – Customers
  – Users
  – Domain experts

• Must identify the *stakeholders*
What vs how

- This distinction is inaccurate
- “What” only captures *functional requirements*
- There are also *nonfunctional requirements*
Nonfunctional requirements

• Look and feel
  – The appearance
• Usability
• Performance
• Maintainability
• Portability
• Operational
  – The operating environment of the product
• Security
• Legal
  – What laws and standards apply to the product
• Cultural and political
Main “components” of (functional) requirements

• The application domain
• The problem to be solved
• The machine to build (the software solution)
The application domain and the machine

• Application domain
  – set of observable phenomena and properties

• Machine
  – set of implementation phenomena and properties

• Some phenomena are *shared*
  – they are the interface between application domain and machine
An abstract view

shared phenomena
(connection domain)

the application
domain

requirements

the machine
Sample shared phenomena

• Keystrokes typed by user and received by software
• Signal sent by microprocessor to car’s fuel injectors
• Directory in which an existing program puts files retrieved by another program to be designed
Sample phenomena

the application domain

the machine

Wheels_turning
Reverse_enabled
Moving_on_runway
Wheel_pulses_on
Connection domains

Example: info system to report on temperatures around the world
The goal (requirement)

• Give people in the real world the ability to do something they can’t do
• The machine interacts with the real world to achieve the goal

(in some cases the goal of the machine to build is to interact with an existing machine)
Context diagrams

• They describe the separation between machine and environment
• They describe all the involved domains, relevant to the problem requirements
• Connections show the existence of shared phenomena
Patient monitoring

Patients in an intensive-care ward in a hospital are monitored by electronic analog devices attached to their bodies by sensors of various kinds. Through the sensors the devices measure the patients’ vital factors: one device measures pulse rate, another blood pressure, and so on. A program is needed to read the factors, at a frequency specified for each patient, and store them in a database. The factors read are to be compared with safe ranges specified for each patient, and readings that exceed the safe ranges are to be reported by alarm messages displayed on the screen of the nurse’s station. An alarm message is also to be displayed if any analog device fails.
Context diagram

![Diagram showing Intensive Care Unit connected to Machine]
Domains of interest

• Patients
• Nurse’s station
• Analog devices
  – Because customer’s requirements are expressed in terms of them
    • Patients must be monitored according to specified frequencies and safe ranges
    • The nurse’s station must be notified by alarm messages if anything goes wrong
    • Analog devices must be monitored and any failure must be reported at the nurse’s station
Principle

The Principle of Domain Relevance

Everything that is relevant to the requirements must appear in some part of the application domain
Domains

• What are the entities in the domain?
• What kind of attributes/properties they have?
• What kinds of relationships exist among entities?
• What kinds of events may occur?
Detailed context diagram

- Analog Devices
- Patients
- Machine
  - Safe Range &Freq. Specifiers
- Database
- Nurse’s Station
Context diagrams: containment

- Observable phenomena satisfy some laws that depend on the *domain*. For example:
  - Wheels_turning iff Moving_on_runway
  - Wheel_pulses_on iff Wheels_turning
(Functional) requirements

• Describe the problem to solve in terms of the problem’s domain
• State the desired properties that must be established by the machine
• Can be formalized as predicate on observable phenomena in the application domain
• Predicates may involve both shared and non-shared phenomena
  – Reverse_enabled iff Moving_on_runway
Requirements vs. specification

• Specifications are derived from requirements, using domain knowledge
• State the properties of the behavior that must be established at the interface between the machine and the external environment (application domain)
• They are expressed in terms of the shared phenomena
Specifications

• They are special kinds of requirements that mention only shared phenomena (see slide 12)

• The phenomena constrained by the specification are those under control of the machine to build

• Example
  – Reverse_enabled iff Wheels_pulses_on
Requirements specification activity

• Deals with 3 parts
  – G the problem (requirements)
    • the goal we want to achieve (expressed in terms of observable phenomena, not necessarily shared)
  – D the domain
    • phenomena and properties of the world of interest
  – S the domain/machine interface (specification)
    • shared phenomena and properties to be enforced by software

• Required reasoning
  – D ∧ S ⊨ G
Warning on terminology

• No established and accepted terminology
• Requirement, specification, design are terms used with different meanings
• Often because they are used at different stages of development
Towards a common terminology

• Requirement for a “service” provided by X expressed by Y
  – Property expressing Y’s expectation from the availability of X

• Specification
  – Contract between X, which provides the service, and Y, which “consumes” the service

• Design
  – To provide structure
In our specific context

• Requirements
  – The required effects of an application on its external environment

• Specification
  – Constraints on the interface between the external environment and the machine

• Requirements engineering
  – How to design requirements and specifications
Language concepts

• Domain description
  – Designations
  – Definitions
  – Properties
• Requirements
• Specifications
Domain descriptions (1)

• Designations
  – primitive concepts, phenomena of interest
    • Wheels_turning \textit{(for a given airplane)}
    • Wheels_turning(x) \textit{(x is an airplane)}
    • Human(x)
    • …
Domain descriptions (2)

• Definitions
  – just a convenience
  – define new terms (derived concepts) using designated terms
    • brother(x,y) ::= male(x) ∧ ∃ f | father(f, x) ∧ father(f, y)
      ::= is defined as
Domain descriptions (3)

• Properties
  – express our knowledge of reality in indicative mood
  – they are refutable
  – they state something that in principle can be disposed
    • $\forall x, y \ (\text{human}(x) \land \text{mother}(x, y)) \rightarrow$
      
      $\text{(female}(x) \land \text{human}(y))$

      *it is part of the designation the fact that mother(x, y) means that x is mother of y (and not vice-versa)*
Example

\[ \forall i, o, c, t \quad \text{Inflow}(i, t) \land \text{Outflow}(o, t) \land \text{Contains}(c, t) \rightarrow i = c + o \]
Requirements and specifications

• They are *optative* properties
  – They express our wishes
    • *Patients* **must** be monitored according to specified frequencies and safe ranges
    • *The nurse’s station* **must** be notified by alarm messages if anything goes wrong
    • *Analog devices* **must** be monitored and any failure must be reported at the nurse’s station
**Example**

\[
\begin{align*}
\text{Buffer} & : x_i \ x_{i+1} \ \ldots \ x_n \\
\text{In}(t) & : x_1 \ x_2 \ \ldots \ x_n \\
\text{Out}(t) & : x_1 \ x_2 \ \ldots \ x_i
\end{align*}
\]

**Specification**

\[\text{In}(t) = \text{Out}(t) \ \text{cat} \ \text{Buffer}(t)\]
Correctness of requirements specification

• G
  – The problem may not be the real problem

• D
  – The domain description does not reflect reality

• Cannot prove that
  – G follows from D and S

Carlo Ghezzi--SE-Req
Example: what is wrong here?

- D
  - Wheels_turning iff Moving_on_runway
  - Wheel_pulses_on iff Wheels_turning
- S
  - Reverse_enabled iff Wheels_pulses_on
- G
  - Reverse_enabled iff Moving_on_runway
The problem is not the software

- The problem is in domain analysis
  - Wheels_turning iff Moving_on_runway
- This is incorrect, because it can happen that
  - Moving_on_runway ∧ ¬ Wheels_turning
- This is called aquaplaning; it is real case of an accident where a plane overshot the runway on landing
Problem context, domains, subdomains (1)

• What is domain, problem, interface, machine depends on the specific context

• E.g. a realtime OS kernel
  – the application domain is inside the computer

Background processes
Interrupt handlers
Problem context, domains, subdomains (2)

- Distinction can vary from time to time
- Can depend on viewpoint
  - e.g., extension of an information system (IS)
- The existing IS is part of the domain
  (it was the machine to develop in the first place)
An example

Turnstile controlling entrance to the zoo
Domain analysis

• Identify the relevant environment phenomena (here: events)
• “Relevant” with respect to the goals (i.e., control people entrance)

machine controlled

- Push frees turnstile
- Lock
- Unlock electric signals
- Coin coin inserted

shared phenomena

environment controlled
More on shared phenomena

- They are events that occur both in the environment and in the machine
- If a delay were involved, a channel would also be part of our description
More on indicative vs optative descriptions

• Indicative
  – Express facts of the environment (domain knowledge)
    • Can be falsified by observation

• Optative
  – Express our wishes

*Useful to distinguish in textual requirements descriptions*

*The latter describe requirements*
Indicative descriptions

1. Push and Enter alternate
   - One cannot enter without first pushing
   - One cannot push until the previous visitor entered

2. If Lock/Unlock alternate (Unlock first), Push can only occur after Unlock and before Lock

These are real world properties
They do not depend on the machine
Requirements

1. At any time entries should never exceed accumulated payments (for simplicity, assume 1 coin for 1 entrance)
2. Those who pay are not prevented from entering (by the “machine”)

• They are optative descriptions
• Both are said to be safety properties (they state that nothing bad will ever occur)
Deriving specs from requirements (1)

• Must find the constraints on *shared phenomena* to be *enforced by the machine* to achieve the requirements
  – R1 can be enforced by controlling entries or coins
    • the machine cannot compel Coin events; it can prevent Enter events
How to constrain entry events?

• From IND-1 we derive IND-3:
  – At any time t, if e Enter and p Push events were observed, then $p-1 \leq e \leq p$

• From OPT-1 and IND-3 we derive OPT-1a:
  – At any time t, if p Push and c Coin events were observed, then $p \leq c$
How to ensure $p \leq c$?

• When $p = c$, must prevent further Push until Coin event occurs

• But how can the machine prevent Push?
  4. Impose that Lock and Unlock alternate
  5. (Refines OPT-1a)
    i. If locked and $p = c$, the machine must not unlock the turnstile, and
    ii. If unlocked and $p = c$, the machine must perform a Lock in time to prevent further Push
Deriving specs from requirements (2)

Requirement R2

Those who pay are not prevented from entering (by the “machine”)

must be transformed into a specification

6. (Refines OPT-2)
   i. If unlocked and $p < c$, the machine does not lock until there is credit, and
   ii. If locked and there is credit, the machine must perform Unlock event
Specification

4 and 5 and 6

Exercise: Prove that Specs and Env imply Rqmts
Summing up (1)

• Where can we find requirements?
  – in the application domain
• How do we find them?
  – by interacting with the stakeholders
  – by understanding the domain and its properties
• From requirements to specifications
  – we need to understand how the machine-environment interaction is to be constrained to achieve the desirable requirements
Summing up (2)

• From requirements to specification
  – the specification is expressed in terms of the phenomena that are shared between the machine and the environment
  – it must be such that what is requested by the stakeholders (the requirements) are satisfied
Systematic requirements engineering

- Can we find a notation to drive the requirements activity?
- Frame diagrams
  - They extend context diagrams by adding ovals that describe the relationships that specify the requirements
The weather reporting system

The goal of the system is to allow users to express queries about temperatures at different sites around the world.
An image filtering system

The goal of the system is to take an input image as a set of points and produce a filtered output.
Systematic requirements specifications

Can we identify common/recurring schemes? 

(*PROBLEM FRAMES*)

- Support for reusability
- Provide guidance in requirements analysis and specification
The “compiler” frame

Source program (input)

Compiler (the machine)

Executable program (output)

Language and machine semantics

Requirements are expressed as i/o relationships
An example

E-mail filtering system

Filter

Input mail messages

Filter rules

Filtered messages

Usually these problems are purely algorithmic, little interface design (maybe user interface, API to other programs)
The “control” frame

There is no notion of i/o transformation
There are notions of state and state transitions to be controlled

*See case of airplane: Wheel_pulses_on, Reverse_enabled*
The washing machine example

Desired behavior

Controlled domain

Controller

Washing program rules

Washing machine

User interface

Program sequencer
The airplane example revisited

Requirement

Reverse_enabled iff Moving_on_runway

Controller

Program sequencer

Airplane

- Wheels_turning
- Moving_on_runway
  ♦ Wheels_turning iff Moving_on_runway
  ♦ Wheel_pulses_on iff Wheels_turning

Shared phenomena

- Reverse_enabled
- Wheels_pulses_on

denotes containment
The “workpieces” frame

- Request services, provide commands to machine to operate on internal workpieces
  - e.g. word processor, spreadsheet
The text editor example

For these problems, user interface is a predominant aspect of requirements
The “Information System” frame

An IS provides information, in response to requests, about some real-world domain of interest

The real world shares phenomena with the machine which represent entities and domain knowledge that must be represented inside the machine (database, business rules)
Multiframe problems

• Most complex realistic problems require a combination of elementary problem frames
  – They are a multiframe problem
Requirements checking

• Validity.
  – Does the system provide the functions which best support the customer’s needs?

• Consistency.
  – Are there any requirements conflicts?

• Completeness.
  – Are all functions required by the customer included?

• Realism.
  – Can the requirements be implemented given available budget and technology?
Requirements checking

• Verifiability.
  – Is the requirement realistically testable?

• Comprehensibility.
  – Is the requirement properly understood?

• Traceability.
  – Is the origin of the requirement clearly stated?

• Adaptability.
  – Can the requirement be changed without a large impact on other requirements?