Requirements Engineering

by

Luuk Groenewegen

fall 2004

Leiden University, Liacs

course for
Bachelor students in Computer Science

- < 1978: mathematics, operational research
  behaviour modelling and analysis

- >1978: computer science

  software engineering
  information systems

  modelling: statics & dynamics
  aka structure & behaviour

  communication:
  modelling
  better understanding in terms
  of its behavioural effects

  consistency and integration,
  particularly wrt dynamics:
  (behavioural) views
  protocols

  to achieve cooperation / collaboration

resulting in
  coordination specification language
  Paradigm

object-orientation
  combination OMT - Paradigm:
  Socca

architecture / integration of
  software components
  systems: software & organizations

evolution / change of systems,
  possibly on-the-fly

mobility and other aspects,
  particularly wrt dynamics

particular relevance for RE:

- RE is part of software engineering (SE)
  process models of RE

- RE is about software system-to-be:

  specified: modelled from the outside
  typically a declarative model

  fitting in organization:
  business process modelling,
  consistency, integration, coordination

  here lies relation with my research

books:

1. Requirements Engineering, Processes and Techniques; Kotonya, Sommerville; Wiley 98; 0471 97208 8

2. The Unified Modeling Language, User Guide
   Booch, Rumbaugh, Jacobson
   Addison-Wesley 99; 0201 57168 8

or www.omg.org/docs/ptc/03-09-15.pdf
   & www.omg.org/docs/ptc/03-08-02.pdf
   OMG (03): UML 2.0 Infrastructure and Superstructure Specification

3. Fundamentals of Software Engineering
   Ghezzi, Jazayeri, Mandrioli;
   Prentice-Hall 03; 013 099183 X

1: must
2: this or something similar
3: interesting, for general background

set-up book 1:

1. introduction
2. RE process
3. R elicitation and analysis
4. R validation
5. R management
   ---- process until here

   ----technique from here
6. methods for RE
7. viewpoint-oriented R methods
8. non-functional R
9. interactive system specification
10. case study (library)

Note: correspondence title & book structure

Note: UML is not very prominent

course set-up (~ book 1)

1. introduction
   global course structure
   FAQs
   (general) system engineering
   R document

2. RE: requirements engineering
   process models
   actors
   process
   process support
   process improvement

large interlude
   UML 2.0 (in relation to UML 1.4)
   (business) architecture and patterns
      management paradigm
      integration-orientation,
      business process modelling

3. R elicitation and analysis
   processes
   techniques
   prototyping
   model recognition
   simulation

4. R validation
   reviews
   prototyping
   R testing
   model recognition
   simulation

5. R management
   stable & volatile R
   identification and storage
   change management
   traceability
   evolution

7. viewpoint orientation
   VOSE
   VORD
   model views
   consistency

8. non-functional Rs
   classification
   principles
   qualities
   viewpoints
   alignment of business and Rs

what software system should be built

what is the problem?

too many software projects fail
in the sense that
- remain unfinished
- finished but not doing what it should
- finished and doing what it should
but it still doesn't fit in the organization

why is it a problem?

- costs of the effort
- missing benefits of well-aligned software

- software is a technical product but
merely technical specs are insufficient as
-- people don't know what they want
-- people change their mind and wishes

- if you ask them, they don't tell you

- even more misleading,
they might tell you (the) wrong things

moreover, requirements are moving target
as organization changes

- from outside
- from inside
- because of the new software

unhappy consequences

"hard" computer science people complain:
this is too "soft" for us

similarly, RE course is
too soft for computer science students

because of the costs / profits at stake

industry is very keen on improvement
ie better RE

client industry: they really want it
so they are willing to pay

provider industry: they really offer
(small) improvements
as they are willing to sell

what is the / a solution - so far

- software engineering (process), > 1968
separation of concerns in the process
- what?: feasibility, requirements
- how?: architecture, design
- construct: build components, integrate
- deliver
- maintain

+ omnipresent
reviewing, verifying, testing

: quality assurance,
both of product and of process

what is the / a solution - so far

- object-orientation

from 1969 on
programming languages
Simula, Smalltalk, C++,
Eiffel, Java, ...

from 1985 on
modelling languages
ROOM, OMT, UML, ...

connecting / "integrating"
structure - behaviour - communication

and (hopefully) other aspects too

mainly during how?-construction phases

even during what?-phase
by means of "declarative modelling"

why does it help?

postponing technical how?-question
gives more room for
nearly non-technical what?-question

this is why separation of concerns works

integrating features of OO enable
smooth software engineering trajectory
(from less technical to very technical)

why doesn't it help sufficiently?

question remains:
will it fit in organization?

apparently not
unless something extra will be done

tentative solution

make operational (technical)
OO model of business

make declarative (non-technical)
OO model of software system-to-be

study together during RE

why should that help?

question of fitting is being studied
before software system will be made

having eye for flexibility
of both business architecture
and software architecture
probably helps for coping with
later evolution more easily

note

standard objections Computer Sc. community

- people are no software
so OO doesn't apply to
modelling people, business, teams

- one should never use
technical / operational modelling in RE
as RE has a purely declarative character
it addresses "what"
instead of the operational "how"

BUT
- (OO) simulation makes sense
so OO modelling makes similar sense

- the above restriction during RE to
declarative (postponing operational)
makes sense for the software-to-be,
not for the business-as-is

back to problem situation

RE is about:
what software system should be built

compare to SE (software engineering):
how do we produce (concrete) good software
in a good way
ie good product and good production process

in particular RE addresses:
- what should that software do (declarative)
external static specs of product
- how could that software fit (operational)
external dynamic specs of product
- how do we produce these specs (Rs)
static and dynamic specs of
production process of Rs

- feasibility: global idea

- RE: precise specs

- design: how does the product work

- implementation: building the product

- testing: (final) check

- deliverance: introduction, (final) tuning

- maintenance: debugging, extending, changing

study feasibility of product
study feasibility of process
plan production phases
characterize product from outside, in parts
model product, in parts
program product, in parts
integrate characterization parts
integrate model parts
integrate program parts
observe, control, improve characterization
observe, control, improve model
observe, control, improve product
observe, control, improve phases
deliver (in - other - parts)
support
do follow-up

- the above listing does not impose
any ordering between the phases

so, this is not a process model yet

- role of decomposition / composition is
in line with separation of concerns, by
first concentrating on less:
simplification but in all detail
integrating afterwards:
remaining complexity but more global

- possibility for growth via parts

- possibility for theory and exactness via model and its (partial) analysis

- observing, controlling and improving covers
both testing (reviewing, analysing)
and managing

furthermore, combining
observing, controlling and improving of
characterization / model / product
with phases of
deliver / support / do follow-up
possibly from well-chosen viewpoints
can result in
welcome, additional insight into
the use of the system
as it - then - is
as it should be
as it could be

here we see how the software system
both statically and dynamically
influences the surrounding business

this is relevant for
external static and dynamic specs
of software (product of RE)

1.1. FAQs: frequently asked questions

requirements: refer to product
properties, ranging from needed to optional;
conditions, circumstances;
services, functionality;

classification into
functional
general usage
system management / administration
embedding / environment

requirements engineering: refers to process
one or more processes, resulting in the Rs;
performing: how to determine
what the software-to-be is supposed to do
and what the role is of that software-to-be

costs: refer to product as well as to process
15% of SE, so 15% of the final software

RE process (model): refers to process
structured set of activities for RE
description / specification comprising:
schedule, roles / responsibilities,
steps to be taken and their I/O,
tool support

ideal RE process?: refers to process
there is no ideal RE process, but

R document: refers to product
document, mainly textual, containing
approved definition or the Rs: the specs

stakeholder: refers to product
all humans relevant for the software-to-be
not only (end) users
but also managers, system administrators,
auditors, specialists

relation Rs - design: refers to process
overlapping, so there is interaction between
the phases of RE and design
dependency relation, traceability:
mainly cause - consequence

Rs management: refers to process
managing changes of Rs

1.2. Systems Engineering

probably better terminology is
Business (Re)Engineering

covers:
- specs for software ie. normal Rs
- specs for hardware
- specs for operations; procedures and processes
in the business, ie. the software environment

wrt the last specs NOTE:
not only for command&control systems
(as the book tells us)

but also for information systems
embedded systems

(so: Business Reengineering indeed)

system engineering is of two types:

(1) so-called
off-the-shelf
component-based
user-configured

ie user does embedding in organization
all specs play their roles,
matching is less deep

(2) so-called
custom system
bespoke system
tailored

ie specialists from SE industry create it

here we see full RE
specs really are engineered

have an open eye for so-called
emergent / overall Rs
vs actual, so-called emergent properties
the system apparently has

those properties only appear (emerge) after

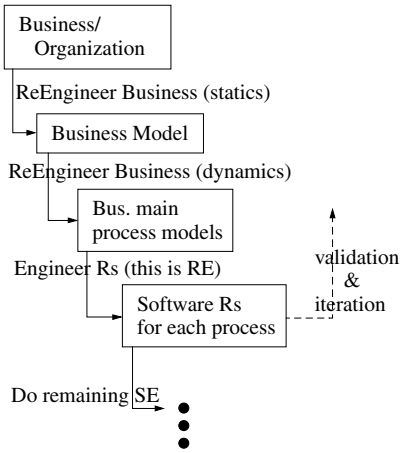
integration of
the software parts

or even later, after
full integration of
software and business

so emergent Rs have a strong tendency to
become clear (too) late too

mostly these Rs belong to so-called
non-functional Rs

adapted version of Fig 1.,
edges correspond to activities:



contains at least

- services / functions
- operational restrictions
- “emergent” Rs as tentative restrictions for the
real emergent properties
- other systems; also business systems
- application domain
- restrictions wrt SE process

there no such thing as a fixed standard
for the R document structure

but guidelines do exist

eg: IEEE guideline for R Doc

1. Introduction
   1.1. purpose R doc
   1.2. scope product
   1.3. definitions, acronyms, abbreviations
   1.4. references
   1.5. overview R doc

2. General Description
   2.1. product perspective (role)
   2.2. product functions
   2.3. user characteristics / categories
   2.4. general restrictions
   2.5. assumptions and dependencies

note

application domain in 1.2, 2.1 mainly

3. Specific Rs
   3.1. functional
   3.2. non-functional
   3.3. interfaces
   3.4. performance
   3.5. DB Rs
   3.6. design Rs
   3.7. system characteristics
   3.8. quality characteristics

4. Appendices

5. Index

note

- 3.5 without Web access is rather old-fashioned
so: leave room for emerging technologies
- some architecture could be good idea
- required specific dynamic effect on business
should be addressed

categories of R doc users

- R Engineers

- users of software system
many further categories

- (business) managers “above” these users

- managers of SE process (and of RE process)

- software engineers (wrt this system)

- quality assurance engineers

- software maintenance engineers

and in addition

- auditors / controllers / domain specialists
participating as such within the
relevant business processes

writing Rs and R doc:

- understandable for all possible readers
of relevant document part

- dominant role for natural language
drawbacks:
vague; ambiguous; cumbersome
eg.
- complex situations: nesting of
if ... then ... with extra conditions
- sloppy or inconsistent terminology
- default knowledge of
problem domain / technology used

so wrt R doc:
- invest in readability
- take reader’s viewpoint (with less knowledge)
- invest in review and improvement iteration