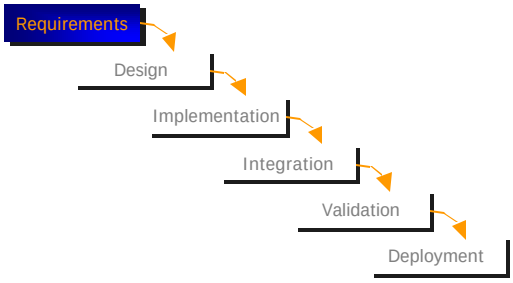
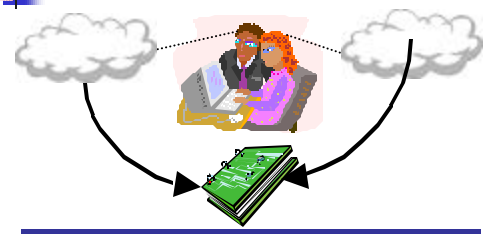
Software Requirements

David S. Rosenblum
ICS 221
Winter 2001

Software Lifecycle Context (Waterfall Model)



The Requirements Engineering Problem



Difficult even when a working implementation exists! (Heninger/A-7)

Goals and Objectives of a Requirements Specification

- Understand and specify customers' needs
  - customer knows best what is wanted but usually doesn't know what can be achieved
- Identify functional capabilities to be provided
- Identify non-functional and environmental constraints to be satisfied
- Develop a contract between customer and developer
- Provide basis for definitive validation

Goals and Objectives (Heninger)

- Specify external behavior only
- Specify constraints on the implementation
- Be easy to change
- Serve as a reference tool
- Record forethought about system's lifecycle
- Characterize acceptable responses to undesired events

So how do you decide what does and does not go into a requirements specification?

The Traditional View: What vs. How

*A software requirements specification should describe **what** a system is supposed to do without discussing **how** it is going to do it*

- Rationale:
  - Inclusion of design and implementation detail ...
    - ... stifles the creativity of the designer and programmer
    - ... forgoes the expertise of the designer and programmer
    - ... biases the statement and understanding of the problem
    - ... potentially restricts the nature of the delivered system

Example: Requirements of a Sort Routine (I)

- What:
  - The sort routine must take a list of numbers as input and return the ordered list as output
- How:
  - Bubble sort
  - Insertion sort
  - Quicksort
  - AKS sort
  - Ideal (constant-time) sort
  - ...

Example: Requirements of a Sort Routine (II)

- Previous formulation of "what" was both too specific and too vague
  - Why just numbers?
  - What is "the ordered list"?
- Try #2:
  - The sort routine must take as input a list of elements all of the same type and return the same list of elements as output according to the order relation defined for that type

Example: Requirements of a Sort Routine (III)

- Formulation #2 of "what" still doesn't consider all possibilities
  - What if the input is empty?
  - What if the type does not define a total order?
  - What if the type defines multiple orders (e.g., ascending and descending) or a cyclic order (e.g., rock/scissors/paper)?

Example: Requirements of a Sort Routine (IV)

- Try #3:
  - The sort routine must take a list of elements of the same totally-ordered type as input and return the same list of elements as output according to some order relation defined for that type
  - The sort routine must return the null list as output if given the null list as input
- Also, isn't "type" really a programming language concept???

Formal Specification of Requirements

Many formal theories, notations and languages have been developed for formal specification of requirements

- Advantages
  - Precise, unambiguous statement of "contract"
  - Can be mathematically analyzed for early detection of faults, internal inconsistencies and ambiguities
  - Can be used to mathematically prove consistency of design with requirements
- Disadvantages
  - Requires mathematical sophistication of supplier *and* customer
  - Difficult to scale, organize
  - Typically need "multi-paradigm" language

Example: Z Calculus

- Popular formal notation based on simplified set theory
  - Specifications organized into *schemas*
  - Schemas composed and related according to a special *schema calculus*
- Has been used on many large systems
  - Reengineering of CICS by IBM in UK
  - "8.9% productivity improvement"
- Z culture eschews automation and tools
  - Natural language commentary an essential part of a Z description

A Z Specification of a Sort Routine (I)

ElemProps

$\_ \leq \_ : ELEM \leftrightarrow ELEM$

$permutations : \text{seq}\, ELEM \leftrightarrow \text{seq}\, ELEM$

$ordered : \mathbb{P}\, \text{seq}\, ELEM$

$\forall e : ELEM \bullet$
$\quad e \leq e .$

$\forall e_1, e_2 : ELEM \bullet$
$\quad e_1 \leq e_2 \lor e_2 \leq e_1 .$

$\forall e_1, e_2 : ELEM \bullet$
$\quad e_1 \leq e_2 \land e_2 \leq e_1 \Rightarrow e_1 = e_2 .$

$\forall e_1, e_2, e_3 : ELEM \bullet$
$\quad e_1 \leq e_2 \land e_2 \leq e_3 \Rightarrow e_1 \leq e_3 .$

$(\emptyset, \emptyset) \in permutations .$

$\forall s_1, s_2 : \text{seq}\, ELEM \bullet$
$\quad (s_1, s_2) \in permutations \Leftrightarrow \exists e_1, e_2 : ELEM \bullet$
$\quad\quad e_1 \in s_1 \land e_2 \in s_2 \land e_1 = e_2 \land (s_1 \setminus \{e_1\}, s_2 \setminus \{e_1\}) \in permutations .$

$\emptyset \in ordered .$

$\forall s : \text{seq}\, ELEM \bullet$
$\quad s \in ordered \Leftrightarrow \forall i \in \{1 \,..\, \#s - 1\} \bullet$
$\quad\quad s(i) \leq s(\#s) \land s \setminus \{s(\#s)\} \in ordered .$

A Z Specification of a Sort Routine (II)

Sort

ElemProps

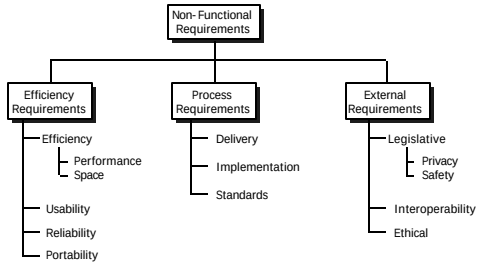
$in : \text{seq}\, ELEM$

$out : \text{seq}\, ELEM$

$(in, out) \in permutations .$

$out \notin ordered .$

Non-Functional Requirements (Somerville)



Another Example: A Telephone Switch (I)

- What:
  - When a caller picks up a telephone receiver, the switch will emit a *dial tone*
  - This dial tone will be a polyphonic composition of two sustained tones, one of frequency 183 Hz (F# below middle C) and the other of frequency 458 Hz (A# above middle C)
  - The switch will emit the dial tone within 500 milliseconds after the caller picks up the receiver

Another Example: A Telephone Switch (II)

- What (continued):
  - The switch will emit this dial tone until either the caller dials a digit or the switch has waited 20 seconds
  - If the switch emits the dial tone for 20 seconds, then it will replace it by a *fast busy signal* that it will emit until the caller hangs up
  - The fast busy signal will be an annoying tone that the switch repeats every 750 milliseconds, with each repetition lasting 600 milliseconds followed by a period of silence lasting 150 milliseconds
  - ...

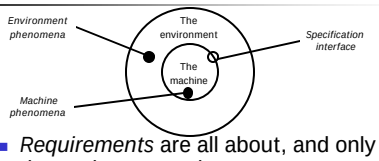
Another Example: A Telephone Switch (III)

- Which of these requirements are functional and which are non-functional?
- There's a lot of "how" in those requirements
  - Rationale: One person's "what" is another person's "how" (Alan Davis)
  - But this distinction is unsatisfying, as are
    - Analysis vs. Design
    - Logical vs. Physical
    - External vs. Internal
    - Conceptualization vs. Realization

A Clearer View (Michael Jackson)

- A software system is a *machine* introduced into the *world* to have some effect there
- The parts of the world that affect the machine and are affected by it are its *application domain*
- The *problem context* is the part of the world in which the machine will be installed and its effects and benefits felt and evaluated
- The interaction between the world and the machine is characterized in terms of their *shared phenomena*

Requirements According to Jackson



- *Requirements* are all about, and only about, the environment phenomena
- *Programs* are all about, and only about, the machine phenomena
- *Specifications* are all about, and only about, the shared phenomena at the environment/machine interface

What vs. How According to Jackson

- *What* the system does must be explained only in terms of the application domain (i.e., in terms of phenomena in the world)
- *How* the system does it must be explained only in terms of the machine

Example: Requirements for an Alarm Clock

- Bad: "The clock displays the wakeup time according to presses of its *hour* and *minute* buttons. When the clock determines that the current time equals the wakeup time, it emits an alarm tone until the alarm is switched off."
  - This is all about the clock (the machine)
- Good: "A person presses buttons to select a wakeup time. At the wakeup time, the person is presented an alarm tone until the person presses a button to disable it."
  - This is all about the user of the clock and the interface to the machine (the world)

Specification of the Person/Clock Interface (I)

- The specification must fully describe the following shared phenomena:
  - The button presses through which the person *selects* and the clock *receives* the wakeup time
  - The button presses by which the user *indicates* and the clock *detects* enabling and disabling of the alarm
  - The alarm tone, which the user *hears* and the clock *emits*
  - The clock display, by which the user *sees* and the clock *displays* the time
  - ...

Specification of the Person/Clock Interface (II)

- The programmer implements the machine according to the specification
- The specification is derived from an analysis of the requirements

The A-7 Experience (Heninger) (I)

- Early example of a systematic approach to requirements engineering
- Semi-formal, tabular expression of state-based requirements
  - Exploited for clarity of communication, not automation of analysis
- An excellent example of expressing requirements in terms of phenomena of the environment

The A-7 Experience (Heninger) (II)

- Much requirements engineering work has built on and formalized the A-7 work
  - Model-checking state-based requirements (Atlee & Gannon)
  - Requirements analysis for safety-critical systems (Heimdahl and Leveson)

More Recent Trends

- Goal-directed requirements and scenarios
  - van Lamsweerde et al.
- Techniques for managing requirements and viewpoints
  - Anthony Finkelstein, Bashar Nuseibeh et al.
- Semantic models for domain descriptions
  - Michael Jackson and Pamela Zave
- The Feature Interaction Problem

For Further Reading (I)

- Books
  - Michael Jackson (the other one), *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, ACM Press/Addison-Wesley, 1995.
- Journals
  - IEEE Transactions on Software Engineering
  - ACM Transactions on Software Engineering and Methodology

For Further Reading (II)

- Conference Proceedings
  - Int'l Conference on Software Engineering (ICSE)
  - ACM SIGSOFT Int'l Symposium on the Foundations of Software Engineering (FSE)
  - European Software Engineering Conference (ESEC)
  - IEEE Computer Society Int'l Symposium on Requirements Engineering