Introduction to Software Engineering

ECSE-321

Unit 6 – Requirements & Specification

Requirements Engineering

- Hardest part of building a software system is deciding what to build!
  - cripples the process if done wrong
  - costly to rectify later
- Goal of requirement engineering is to determine (pick one)
  - what software the client ***wants***?
  - what software the client ***needs***?

Requirements Engineering...

- A ***requirement*** is an expression of desired behavior
- A requirement deals with
  - objects or entities
  - the state they can be in
  - functions that are performed to change states
- Requirements focus on the customer needs, not on the solution
  - designate *what* behavior, without saying *how* that behavior will be realized

Why Are Requirements Important?

- Top factors that caused project to fail
  - Incomplete requirements
  - Lack of user involvement
  - Unrealistic expectations
  - Lack of executive support
  - Changing requirements and specifications
  - Lack of planning
  - System no longer needed
- Some part of the requirements process is involved in almost all of these causes

Determining Stakeholders and Needs

- Must determine stakeholders
  - Anyone who benefits from the system designed
  - E.g., who’s client and who’s user?
- Try to understand what their needs are
- Reconcile different needs/points of view

Determining Stakeholders

- **Clients**: pay for the software to be developed
- **Customers**: buy the software after it is developed
- **Users**: use the system
- **Domain experts**: familiar with the problem that the software must automate
- **Market researchers**: conduct surveys to determine future trends and potential customers
- **Lawyers or auditors**: familiar with government, safety, or legal requirements
- **Software engineers** or other technology experts

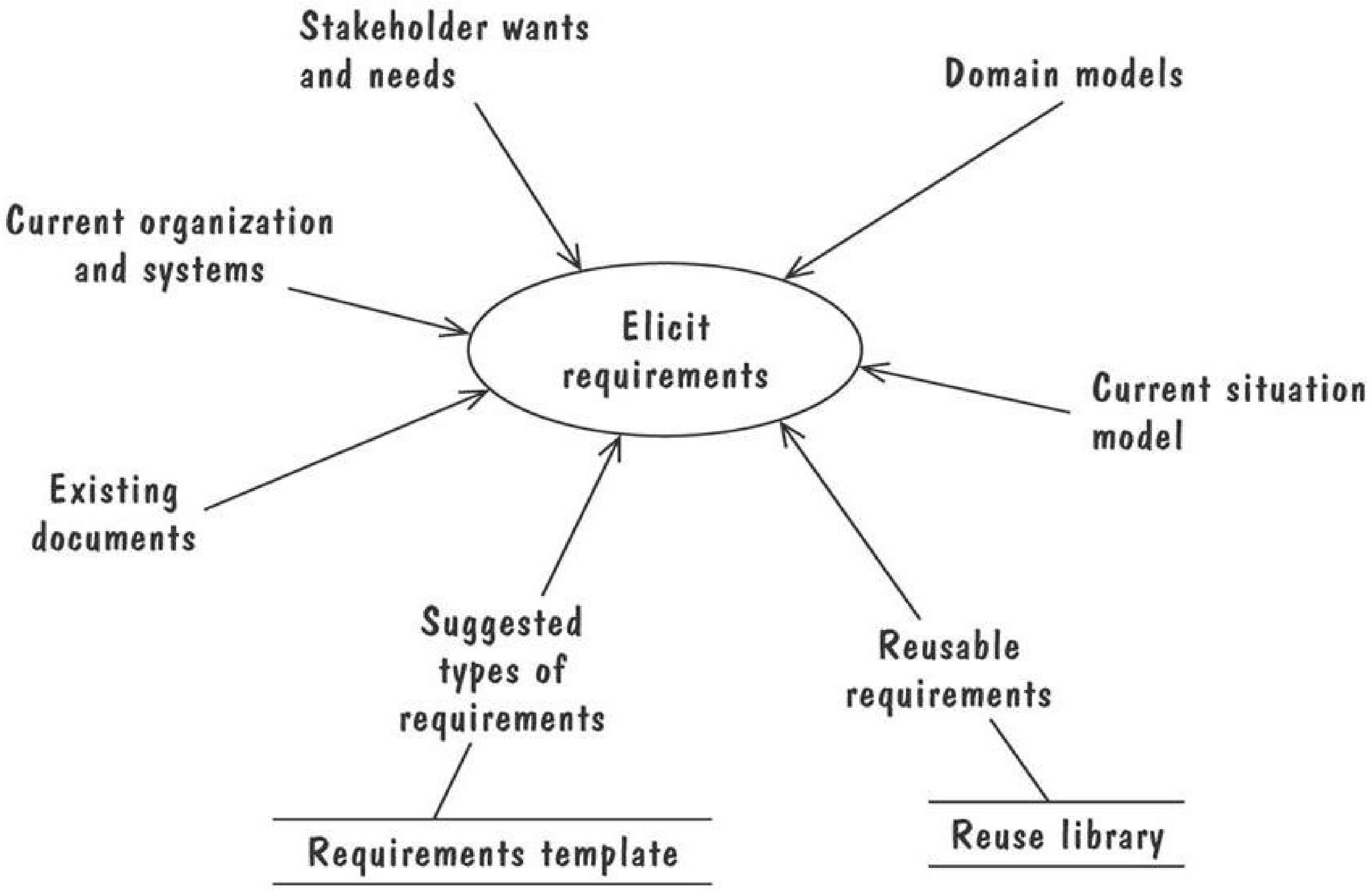
Requirements Elicitation from Stakeholders

- Customers do not always undertand what their ***needs*** and problems are
- Important to discuss the requirements with everyone who has a stake in the system
- Come up with agreement on what the requirements are
  - If we can not agree on what the requirements are, then the project is doomed to fail

Means of Eliciting Requirements

- Interviewing stake holders
- Reviewing available documentations
- Observing the current system (if one exists)
- Apprenticing with users to learn about user's task in more details
- Interviewing users or stakeholders in groups
- Using domain specific strategies, such as Joint Application Design
- Brainstorming with current and potential users

Means of Eliciting Requirements...



Interviewing

- One approach is obvious
  - sit down with client/user and ask questions
  - listen to what they say and what they don’t say
- Less obvious approach
  - master-apprentice relationship
  - have them teach you want they do
  - go to workplace and watch them do the task
- All types of interviews get details
  - ask for copies of reports, logs, emails on process
  - these may support, fill in, or contradict what the user said

Disadvantages of Talking

- Interviews are useful, but..
  - “communication gap” can prevent actual requirement elicitation
  - communication could be misunderstood due to different vocabulary
  - users/clients not knowing enough about computer science to know what is possible
  - or what is impossible
- Idea: better to gather requirements in multiple ways

Strawmen

- Convey the idea without writing any code
- Could help in user/client interviews
- Sketch the product for the user/client
  - Flowcharts
  - HTML mock-ups
  - Illustrate major events/interfaces/actions

Rapid Prototyping

- Write a prototype
  - major functionality, superficially implemented
  - many “corner cases” not handled
- Show prototypes to the user/client
  - users have a real system – more reliable feedback (e.g., mock up model homes)
  - refine requirements
  - significant investment

Pitfalls of Rapid Prototyping

- Needs to be done quickly
  - Remember, this is requirements phase
  - Danger of spending too long refining the prototype
- Prototype becomes the product
  - prototypes not very thoroughly analyzed
  - product can inherit the sub-optimal architecture
- Prototype serves as the spec
  - prototype is incomplete, may be even contradictory
- If done right, can be extremely useful

Techniques for Requirements Elicitation

- Find out what users/clients need
  - not necessarily what they say they want
- Use
  - Interviews
  - User stories
  - Strawmen
  - Rapid prototyptes
- Other appropriate mechanisms to elicit requirements

Using Viewpoints to Manage Inconsistency

- No need to resolve inconsistencies early in the requirements proces
- Stakeholders' views documented and maintained as separate ***Viewpoints*** through the software development process
  - The requirements analyst defines consistency rules that should apply between Viewpoints
  - The Viewpoints are analyzed to see if they conform to the consistency
- Inconsistencies are highlighted but not addressed until there is sufficient information to make an informed decision

Resolving Conflicts – Viewpoints Not Sufficient

- Different stakeholder has different set of requirements
  - potential conflicting ideas
- Need to prioritize requirements
- Prioritization might separate requirements into three categories
  - *essential*: absolutely must be met
  - *desirable*: highly desirable but not necessary
  - *optional*: possible but could be eliminated

Characteristics of Requirements

- Correct
- Consistent
- Unambigious
- Complete
- Feasible
- Relevant
- Testable
- Traceable

Types of Requirements

- **Functional requirement**: describes required behavior in terms of required activities
- **Quality requirement** or **nonfunctional requirement**: describes some quality characteristic that the software must possess

Types of Requirements...

- **Design constraint**: a design decision such as choice of platform or interface components
- **Process constraint**: a restriction on the techniques or resources that can be used to build the system

Making Requirements Testable

- Fit criteria from objective standards for judging whether a proposed solution satisfies the requirements
  - It is easy to set fit criteria for quantifiable requirements
  - It is hard for subjective quality requirements
- Three ways to help make requirements testable
  - Specify a quantitative description for each adverb and adjective
  - Replace pronouns with specific names of entities
  - Make sure that every noun is defined in exactly one place in the requirements documents

Specifications

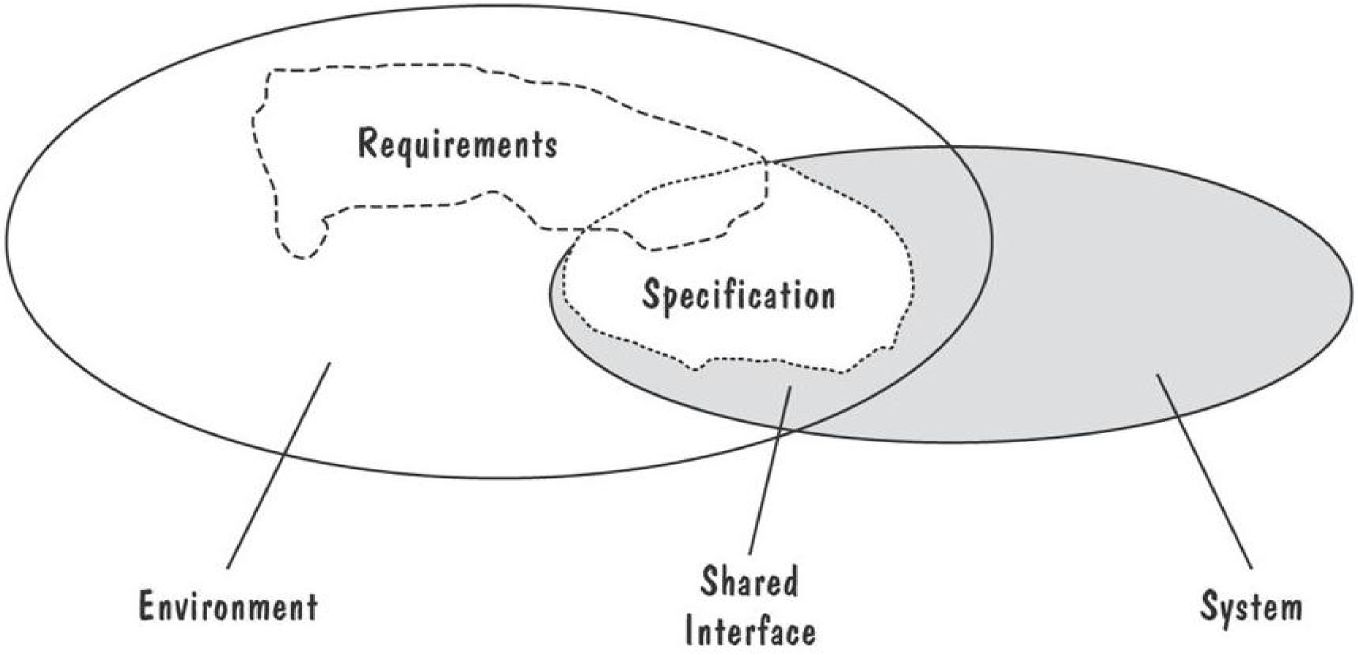
- Describe the functionality of the product
  - Precisely
  - Covering all circumstances
- Move from the finite to infinite
  - Finite examples (requirements) to infinite set of possible computations
  - not easy

Two Kinds of Requirements Documents

- R**equirements definition**: a complete listing of everything the customer wants to achieve
  - Describing the entities in the environment where the system will be installed
- **Requirements specification**: restates the requirements as a specification of how the proposed system shall behave

Two Kinds of Requirements Documents...

- Requirements defined anywhere within the environment's domain, including the system's interface
- Specification restricted only to the intersection between environment and system domain



Different Views of Specifications

- Developers
  - specifications must be detailed enough to be implementable
  - unambiguous
  - self-consistent
- Client/user
  - specifications must be comprehensible
  - must not be too technical
- Legal
  - specification can be a contract
  - should include acceptance criteria; if the software passes X, Y, and Z it will be accepted

Informal Specifications

- Written in natural language
  - e.g., in English
- Example
  - “if sales for the current month are below the target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of the difference between target sales and actual sales the previous month, or if the difference between the target sales and actual sales for the current month is less than 5%

Problem with Informal Specification

- Informal specifications of any size can suffer from serious problems
  - Omissions – missing elements
  - Ambiguities – something that can be interpreted in multiple ways
  - Contradictions – spec says “do A” and also “don’t do A”
- *These problems can plague the software unless detected and fixed in the spec.*

Comments on Informal Specification

- Informal specification is universally reviled
  - By academics
  - By “how to” authors
- Informal specifications are also widely practiced!
  - Why?

Why do People Use Informal Specs

- The common language is natural language
  - Customers can’t read formal specs
  - Neither can most programmers!
  - Or most managers
  - A least common denominator effect takes hold
- Truly formal specs can be time consuming
  - And relatively hard to understand
  - And overkill for most projects

Semi-Formal Specs

- Best current practise is “semi formal” specs
  - Allows more precision than natural language where desired
- Usually boxes and arrows notation
- Pay attention to
  - what boxes mean
  - what arrows mean
  - different in different systems

Stating Requirements – Pitfalls and Tips

- Suppose you have a requirement stating "Operation A will occur and operation B will occur, or operation C will occur."
- Following questions:
  - Operation A occurs before B? Does order matter?
  - Must Operation C occur only if neither A nor B occur?
  - May Operation C occur even if both A and B occur?

Stating Requirement – Pitfalls and Tips

- Should following the rules for good technical writing:
  - write complete, simple sentences in active voice
  - define terms clearly and use them consistently
  - use the same word for a concept – avoid synonyms
  - provide a table of contents
  - use tables, lists, white space, and other formatting aids
  - leave margins ragged on right

Stating Requirements – Pitfalls and Tips

- Software product *must* have certain features, function, and capability
  - ***Express all requirements using the words "must" or "shall"***
- Example
  - The product must display all results to three decimal places.
  - The product will display all results to three decimal places.

Stating Requirements – Pitfalls and Tips

- Specifications should be ***verifiable***
  - *The product must produce reports in an acceptable amount of time.*
  - The product must produce reports in five minutes or less from the time the report is requested.
  - The product user interface must be user-friendly.
  - Eighty percent of first-time users must be able to formulate and enter a simple query within two minutes of starting to use the program.

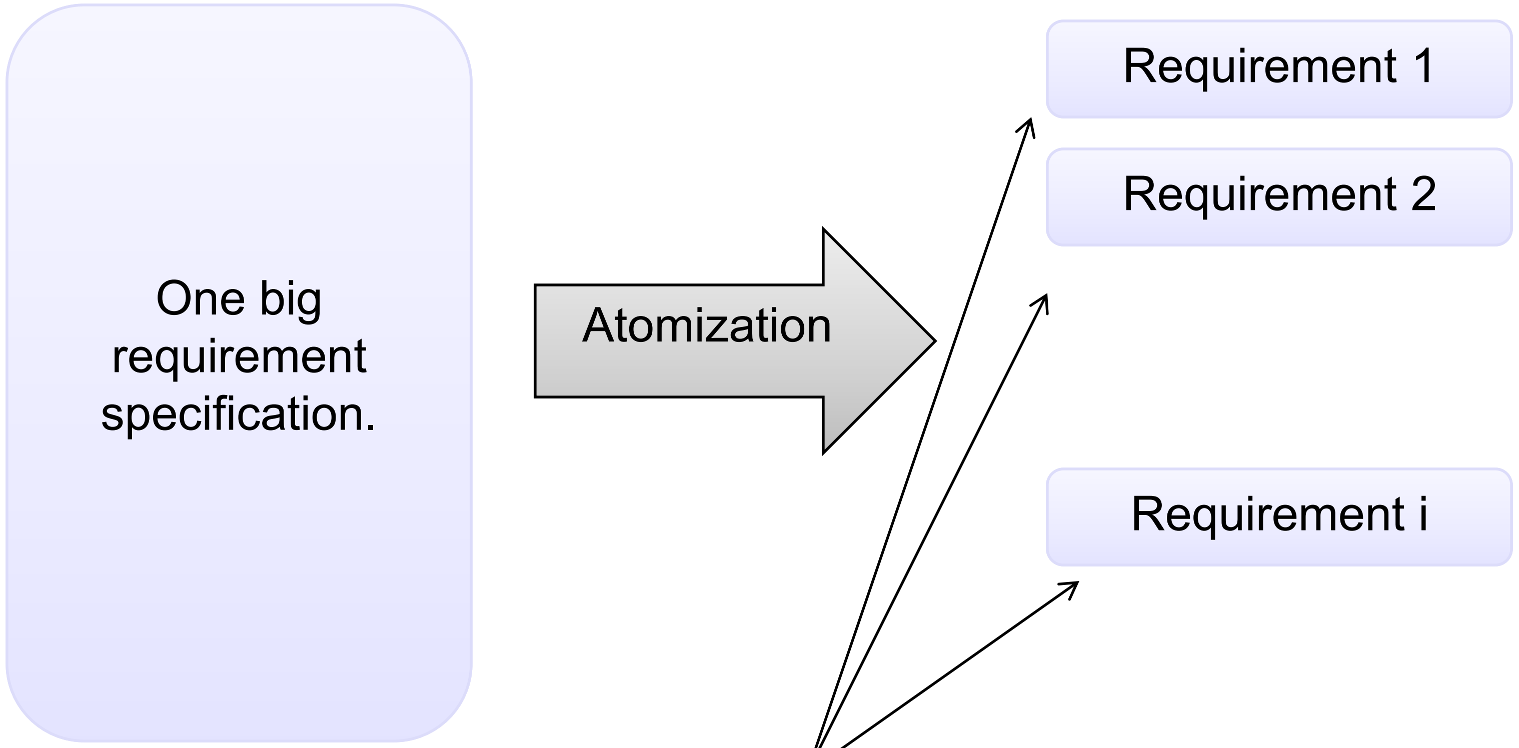
Stating Requirements – Pitfalls and Tips

- Verifiable requirements are ***crucial***
- Verifiable requirements capture the needs to satisfy the clients
- ***Always write verifiable requirements.***

Stating Requirements – Pitfalls and Tips

- ***Requirements Traceability***
  - Ability to track requirements from their expression in an SRS to their realization in engineering design documentation, source code, and user documentation and their verification in reviews and tests.
- Requirement traceability relies on
  - isolating and identifying individual requirements
  - parts of a design, code, test case can be correlated to the relevant portions

Stating Requirements – Pitfalls and Tips



Easier to perform requirement tracing on the ***atomic*** requirements.

Requirement Atomization

- Example:
  - Staff must be able to add computers to the tracking system. When a computer is added, the tracking system must require the staff member to specify its type and allow the staff member to provide a description. Both these fields must be text of length greater than 0 and less than 512 characters.

    The tracking system must respond with a unique serial number required for all further interactions with the tracking system about the added computer.

Requirement Atomization

1. The tracking system must allow staff to add computers to the system.

   1.1 When a computer is added, the system must require the staff to provide type data for the added machine.

   1.1.1 A computer's type data must be text of length greater than 0 and less than 512

   1.2 When a computer is added to the system, the system must allow staff to provide a description data.

   1.2.1 A computer's description date must be text of length between 0 and 512.

   1.3 The tracking system must respond to added computer input with a unique serial number identifying the computer in the tracking system.

   1.4 All further interactions between staff members and the tracking system about a computer must use the unique serial number assigned by the tracking system.