Understanding Requirements
Requirements engineering remains one of the most problematic aspects of software-intensive systems development.

“So what’s all the hub-bub,... bub?”
Bugs Bunny (1940 - )
US Cartoon Character
Difficulties of Requirements

What is so hard about requirements?

• Everything!!!
  – Finding requirements
  – Writing down requirements
  – Measuring compliance
    • Verification
    • Validation
"The hardest single part of building a software system is deciding what to build....No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

1. A condition or capability needed by a user to solve a problem or achieve an objective

2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents

3. A documented representation of a condition or capability as in (1) or (2)
• A requirement is a clear description of
  – the purpose the software is to serve
  – what the software must do to serve that purpose

• Raw requirements are often written by and/or from the standpoint of users
  – function oriented
  – informal and incomplete
  – must be refined
Constraints

• Designers and implementers do not have complete freedom to create as they please.
  – pre-made design decisions
  – rules, standards
  – budget, schedule

• These are constraints on how designers and implementers may satisfy the requirement.
  – constraints limit choices
Types of Requirements

• Functional (what)
  – “Input X produces Y”

• Non Functional (Christel and Kang)
  – Quality Attributes
  – Interfaces
  – Design constraints

• Implied (Very Nebulous….)
REMEMBER, QUALITY IS OUR TOP PRIORITY.

QUESTION: IS IT MORE IMPORTANT THAN SAFETY?

OOH...I FORGOT ABOUT THAT ONE.

QUESTION: IS QUALITY MORE IMPORTANT THAN OBEYING THE LAW?

WELL, PROBABLY NOT.

IF WE COULD MAXIMIZE SHAREHOLDER VALUE BY SELLING LOWER QUALITY ITEMS...

...WOULDN'T WE HAVE A FIDUCIARY RESPONSIBILITY TO DO IT?

I'M SURE IT'S IN THE TOP FOUR.

WHAT IF WE HAD TO LIE TO ACHIEVE QUALITY?
Non-Functional Quality Attributes

- Performance
- Modifiability
- Reusability
- Reliability
- Stability
- Security
- Extendibility
- Portability
- Usability
  - User friendly?
- Scalability
- Data integrity
- & more

Recent Favorites: “Wowability” & “Buildability”
• Systematic way of getting from need to specification – *must be planned*
  – elicit need
  – analyze
    • validate and quantify functional, quality attribute requirements, and constraints
    • set completion criteria
    • establish working agreement (*Statement of Work*)
  – document
• **Inception**—ask a set of questions that establish …
  – basic understanding of the problem
  – the people who want a solution
  – the nature of the solution that is desired, and
  – the effectiveness of preliminary communication and collaboration between the customer and the developer

• **Elicitation**—elicit requirements from all stakeholders

• **Elaboration**—create an analysis model that identifies data, function and behavioral requirements

• **Negotiation**—agree on a deliverable system that is realistic for developers and customers
• **Specification**—can be any one (or more) of the following:
  – A written document
  – A set of models
  – A formal mathematical
  – A collection of user scenarios (use-cases)
  – A prototype

• **Validation**—a review mechanism that looks for
  – errors in content or interpretation
  – areas where clarification may be required
  – missing information
  – inconsistencies (a major problem when large products or systems are engineered)
  – conflicting or unrealistic (unachievable) requirements.

• **Requirements management**
Inception

• Identify stakeholders
  – “who else do you think I should talk to?”
• Recognize multiple points of view
• Work toward collaboration
• The first questions
  – Who is behind the request for this work?
  – Who will use the solution?
  – What will be the economic benefit of a successful solution
  – Is there another source for the solution that you need?
Eliciting Requirements

• meetings are conducted and attended by both software engineers and customers
• rules for preparation and participation are established
• an agenda is suggested
• a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
• a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
• the goal is
  – to identify the problem
  – propose elements of the solution
  – negotiate different approaches, and
  – specify a preliminary set of solution requirements
Requirements Elicitation: Job #1

• If you do not meet the needs of the true users, your product will fail.

• The first task of requirements elicitation is to identify those users.

• If you do not have access to the true users, you will not determine their needs.

• It is risky to do requirements elicitation at arm’s length or through intermediaries.
Eliciting Requirements

- Conduct FAST meetings
- Make lists of functions, classes
- Make lists of constraints, etc.
- Use QFD to prioritize requirements
- Informally prioritize requirements
- Create Use-cases

- Define actors
- Write scenario
- Complete template
Quality Function Deployment

• Function deployment determines the “value” (as perceived by the customer) of each function required of the system
• Information deployment identifies data objects and events
• Task deployment examines the behavior of the system
• Value analysis determines the relative priority of requirements
Elicitation Work Products

- a statement of need and feasibility.
- a bounded statement of scope for the system or product.
- a list of customers, users, and other stakeholders who participated in requirements elicitation.
- a description of the system’s technical environment.
- a list of requirements (preferably organized by function) and the domain constraints that apply to each.
- a set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- any prototypes developed to better define requirements.
Problems

• Some major issues encountered when you elicit requirements include
  – inarticulateness
  – terminology
  – hidden assumptions
  – preconceived solutions

“You don’t know what you don’t know.”
Many stakeholders (especially users) cannot explain what they do or what they need. They
– remember the exception, and forget the routine
– underemphasize the prominence of simple stuff
– focus on what doesn’t work, not what does work

One articulate user can mislead other stakeholders and build a false consensus.
Inarticulateness – 2

• Quality attribute requirements are the most difficult for stakeholders to articulate.
  – One word descriptions such as "modifiable" are meaningless.
    • How do you measure modifiability?
    • How important is modifiability with respect to other quality attributes?
  – One stakeholder's modifiability is another stakeholders scalability.
Potential Solutions – 1

- Observation
  - Watch users work (covertly and overtly)
- Interviews with key stakeholders
  - Research interviewee?
- Logging
  - Have users write down what they do as they do it
  - Have them log their time on task
- Develop use case and quality attribute scenarios
  - One describes function one describe quality attribute characteristics
Potential Solutions – 2

• Use cases describe required *functionality*. 
• Quality attribute scenarios describe required *quality attribute properties*.
• Which modifiability requirement is more meaningful?
  – "The system shall be modifiable."
  – "Modify the system to utilize a different COTS discrete event generation package in 12 staff months."
Terminology

- Stakeholders (especially users) have a different vocabulary from that of designers and developers.
  - Special cultures result from special terms used in special ways.
  - Developers must understand terms in the context of the stakeholders and their work
    - critical to understanding stakeholder needs
Potential Solutions

- Domain expert
  - Enlist a domain expert that also has a knowledge of software engineering

- Domain dictionary
  - Build a dictionary of key technical terms and their definition before you elicit requirements

- Domain training
  - Train software engineers in the domain or vice-versa
Hidden Assumptions (Implied)

• The stuff “everybody knows” often goes unstated.
  – The obvious may not be obvious to those lacking domain expertise.
  – A system that violates critical assumptions will fail.

• No matter how obvious, critical assumptions must be explicitly stated and recorded.
Potential Solutions

- Observation (again watch them at work)
- Use Cases
- Role playing
  - have non-experts walk through key use cases and test them for completeness
- Prototypes
- Formal analysis
  - cast requirements into a formal specification that can be rigorously checked for completeness
Preconceiived Solutions

• Some stakeholders think they know the answers to their problems.
  – Sometimes they describe their idea of a solution, not the problem
    • “Just write the code, after all its only pictures,...”
    • “I need a Pentium with,...”
  
• Sometimes they do, but these answers may not be the best.
  – Their ideas may be incomplete, out of date, or wrong
Potential Solutions

• Brainstorming
  – just get everything out on the table; distill raw data – separate problems from solutions

• Causal analysis
  – find out why users want each feature and quality attribute characteristic

• Fantasy
  – invite stakeholders to describe the perfect solution

• Prototypes
Refinement: Analysis

Look for:

- Consistency with objective
- Abstraction vs. detail
- Categorization (triage)
- Bounded and unambiguous
- Specific source (person)
- Conflicts with others
- Achievable
- Testable
Refinement: Negotiation

• Do customers want more than possible (cost, time, scope, quality)
• Prioritize by value and cost
  – Value to the customer
  – Value to other stakeholders?
  – Difficulty to achieve (do the hard first?)
Role of Analysis – 1

• Raw requirements tend to describe a desired product from an unstructured operational perspective such as
  – who will use it
  – what the user would like to have
  – in what context(s) it will be used
  – function and quality attribute necessities

• Unstructured wants and needs must be refined into a requirements specification.
• *Requirements elicitation* is a *divergent* process that gathers more and more data.

• *Requirements Analysis* is a *convergent* process that
  – refines data rather than gathers it
  – structures information
  – prioritizes needs
Role of Analysis – 2

• Each functional requirement, quality attribute, and constraint must be
  – clarified – understandable by all stakeholders
  – quantified – measurable, testable
  – Prioritized
    • According to importance (to which stakeholder)
    • Consideration of difficulty to implement
Clarification – 1

• Each raw requirement must be refined to articulate the need and capture all that is relevant to designers and implementers
  – What is needed?
  – When is it needed?
  – How much of it is needed?
  – How badly is it needed?
  – For how long is it needed?
  – How likely is the need to change over time?
Clarification – 2

• Clarifying and refining requirements may feel a little like elicitation. Clarification
  – requires iteration with the stakeholders
  – may be slow, but should converge

• If you generate lots of new requirements, then you may need to revisit elicitation
  – Do you have the right/same stakeholders?
  – Have any environmental, technological, organizational, or personnel changes occurred?
Quantification

• Raw requirements tend to be unspecific and qualitative
  – Must be able to prove that a product satisfies a requirements

• Requirements specifications must say how big, how much, how fast often, and so forth.
  – If not, they you are setting the stage for failure and disappointment.
• Raw Requirement: “The system shall be intuitively easy to use.”
  – This is un-testable!

• The system interface shall
  – be learnable to 90% proficiency in 2 weeks
  – have an avg. user error rate of less than 2%
  – score at least 85% on a user satisfaction test
Example: Clarification

• Raw Requirement:
  “The system shall be modifiable”
  (You will always lose with this requirement!)

• The system shall accommodate
  – changes in the user interface without impact to other elements of the system
  – changes to element X in Y staff hours
Priorities

• Some requirements are more important than other requirements
  – some functionality is urgently needed
  – some quality attributes are essential
  – some requirements are hard to achieve

• Prioritization in the specification is essential for
  – setting expectations, reasoning about technical tradeoffs, planning the work
• Involve stakeholders in prioritization
  – Quality Attribute Workshop (QAW)

• Keep it simple…get creative…

"The system shall respond to external interrupts in 3ms."
"The system shall display warning messages in red."

<table>
<thead>
<tr>
<th>Stakeholder votes</th>
<th>Critical</th>
<th>Important</th>
<th>Don't Care</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>20</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Warning Messages</td>
<td>0</td>
<td>2</td>
<td>30</td>
</tr>
</tbody>
</table>
Moving Requirements Target

- Stakeholders
  - Provides
  - Basis of
  - Influences

- Functional Requirements
  - Constraints
  - Quality Attributes

- Moving Requirements Target
  - Creates
  - Designs
  - Basis of
  - Influences

- Downstream Designers and Implementers
  - Creates
  - Products
Building the Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram
Use-Cases

• A collection of user scenarios that describe the thread of usage of a system
• Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
• Each scenario answers the following questions:
  – Who is the primary actor, the secondary actor (s)?
  – What are the actor’s goals?
  – What preconditions should exist before the story begins?
  – What main tasks or functions are performed by the actor?
  – What extensions might be considered as the story is described?
  – What variations in the actor’s interaction are possible?
  – What system information will the actor acquire, produce, or change?
  – Will the actor have to inform the system about changes in the external environment?
  – What information does the actor desire from the system?
  – Does the actor wish to be informed about unexpected changes?
Use-Case Diagram

- **Arms/disarms system**
- **Accesses system via Internet**
- **Responds to alarm event**
- **Encounters an error condition**
- **Reconfigures sensors and related system features**

- **Homeowner**
- **System administrator**

**sensors**
From the *SafeHome* system …

<table>
<thead>
<tr>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name/id</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>location</td>
</tr>
<tr>
<td>area</td>
</tr>
<tr>
<td>characteristics</td>
</tr>
<tr>
<td>identify()</td>
</tr>
<tr>
<td>enable()</td>
</tr>
<tr>
<td>disable()</td>
</tr>
<tr>
<td>reconfigure()</td>
</tr>
</tbody>
</table>
State Diagram

-- Reading Command --

State name: System status = "ready"
State variables: Display msg = "enter cmd"
State activities: Display status = steady

Entry/subsystems ready
Do: poll user input panel
Do: read user input
Do: interpret user input
Pattern name: A descriptor that captures the essence of the pattern.
Intent: Describes what the pattern accomplishes or represents
Motivation: A scenario that illustrates how the pattern can be used to address the problem.
Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.
Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.
Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.
Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.
Known uses: Examples of uses within actual systems.
Related patterns: One or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.
Negotiating Requirements

• Identify the key stakeholders
  – These are the people who will be involved in the negotiation

• Determine each of the stakeholders “win conditions”
  – Win conditions are not always obvious

• Negotiate
  – Work toward a set of requirements that lead to “win-win”
Validating Requirements - I

• Is each requirement consistent with the overall objective for the system/product?
• Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
• Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
• Is each requirement bounded and unambiguous?
• Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
• Do any requirements conflict with other requirements?
Validating Requirements - II

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?