What is a requirement

- IEEE Standard Glossary of Software Engineering Technology: A requirement is:
  1. A condition or capability needed by a user to solve a problem or achieve and 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 document.
  3. A documented representation of a condition of capability as in 1 or 2.
- User view: What - all about the user’s domain
- Developer view: How - all about the software development domain
The Gap: Customers understand the business domain; Developers the systems development domain

Requirements are an attempt to bridge the Gap
Chaos article factors

<table>
<thead>
<tr>
<th>Success</th>
<th>Challenged</th>
<th>Impaired</th>
</tr>
</thead>
<tbody>
<tr>
<td>user involvement</td>
<td>lack of user input</td>
<td>incomplete requirements</td>
</tr>
<tr>
<td>executive support</td>
<td>incomplete requirements</td>
<td>lack of user input</td>
</tr>
<tr>
<td>clear requirements</td>
<td>changing requirements</td>
<td>lack of resources</td>
</tr>
</tbody>
</table>
Success potential (these factors, weighted, have been used to evaluate and predict success in actual cases)

- user involvement
- exec management support
- clear statement of requirements
- proper planning
- realistic expectations
- small project milestones
- competent staff
- ownership
- clear vision
- hardworking staff
Recommendations

• Lots of milestones
• Iterative development
How do you develop software requirements?

Business Requirements

Vision and Scope Document

System Requirements

Use-Case Document

User Requirements

Quality Attributes

Non-functional Requirements

Constraints

Functional Requirements

Software Requirements Specification
Customer oriented practices can increase the likelihood of a successful outcome

- Proactive approach to communication aimed at
  - mutual understanding
  - clear, timely communication

- Leads to better relationship
  - improved perception of SW development for the customer
  - improved perception of the business issues by the development team
  - increased visibility/transparency

- Functionality, timeliness, and quality that match customer needs
Bridging the Gap: Step1 - Vision and scope document
Develop understanding before committing to requirements

(System development domain)  
(Business domain)

(remember: this is the only reason engineers are needed!)
Add structure to the development *process* that encourages understanding of the important role of the customer

- Budget time and $$ for
  - review meetings
  - client management
  - client monitoring of progress
  - clarifying project goals and requirements
- clearly defined points of
  - contact
  - responsibility

KEY: Customer needs to understand the level of commitment required and the impact of not living up to their commitments BUT think Win-Win
Levels of Requirements

• Business requirements
  ⇒ Project vision and scope
• User requirements
  ⇒ use case or scenario descriptions
• Functional and non-functional requirements
  ⇒ Software Requirements Specification
    ⇒ Development
    ⇒ Testing
    ⇒ QA
    ⇒ Project Management (schedule, budget, etc.)
Risks from inadequate requirements

• Insufficient user involvement
• Feature Creep
• Gold plating
• Minimal specification
• Overlooked use cases
• Inaccurate planning

Can you add to this list? Any actual cases you can discuss?
# Requirements engineering

## Requirements Engineering

<table>
<thead>
<tr>
<th>Requirements development</th>
<th>Requirements management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elicit</td>
<td>Define baseline</td>
</tr>
<tr>
<td>Analyze</td>
<td>Process for change</td>
</tr>
<tr>
<td>Specify</td>
<td>Tracking status</td>
</tr>
<tr>
<td>Verify</td>
<td></td>
</tr>
</tbody>
</table>
Agenda for Labs: Prepare for client meeting (schedule it!)

• Project notebook (journal):
  – Meeting minutes
  – Team makeup etc.
• Think about your approach to gaining a background understanding before your next client interview (detail a process)
• Information sources for competitive intelligence - Web, books
• Create a very preliminary set of questions (think about what you think the answers are:)
  – client’s desired outcome, how this SW differs from what is available
  – what tradeoffs is the client willing to make given time and budget constraints
• Assign action items for each member of the group
Dealing with the client:  
How customers pose risks to project success

Customers
• don’t understand what they want
• won’t commit to written requirements
• insist on new requirements without understanding impacts
• are slow to respond to communication
• will not participate in reviews
• technically unsophisticated
• insist on being involved (inappropriately) in technical details
• don’t understand the process
• are new!!

This is “normal” to some extent even with the “best,” most sincere and diligent clients
Good rapport is easier said than done

- Both sides consider canceling 40% of all out-sourced projects and 65% of all fixed price contracts

Customers
impossible delivery dates
new requirements without additional $$
omitting clear acceptance criteria
inadequate involvement
inadequate visibility

Developers
promising impossible schedules
bidding too low
lacking skills
low quality
missed deadlines
Business and user requirements

**Business Requirements**

**Why**
Guiding framework
product concept
business rationale

Describe objectives that customer wants to achieve or value the system provides

**User Requirements**

**What**
Actual system behavior
Tasks that need to be performed
Non-functional characteristics

Describe with use case and user scenarios
Stages of requirements development

- Business requirements
  ⇒ Project vision and scope
- User requirements
  ⇒ Use case or Scenario descriptions
- Functional and non-functional requirements (distinction? Look!)
  ⇒ Software Requirements Specification
Fact: Requirements change

Build in flexibility -- Even if you don’t use it - it improves resulting design (don’t overdo it! cf Agile/Xtreme philosophy!)

- Design
  - reviews
  - build in time for changes (when appropriate)
- Implementation
  - readable, modifiable requirements, design and code: think about interfaces
  - mini milestones to keep project visible for customer
  - involve the customer in the entire lifecycle model
    - appropriate levels of involvement for interest and ability
Interviewing and questioning techniques

• Be prepared, polite, succinct, diplomatic, and empathetic
  – make the client’s job of helping you as easy as possible
    • make my job of teaching you as easy as possible
      – make it as easy as possible for me to assign you an “A”

• Avoid jargon unless it is the customer’s native tongue
  – document agreed upon definitions that have any relevance to the problem

• Understand who the customer/user is, their area of expertise, responsibility and tailor questions appropriately
  – who is your client, background, interests?
    • with respect!
The Customer-Developer Partnership: Rights and Responsibilities of Software Customers

- Want a collaborative partnership
- Customer Rights -- Developer Responsibilities
- Customer Responsibilities -- Developer Rights
- See Wiegers for details

- Sign-off (our main concern - meet reqts or solve problem?)
  - NOT a way to freeze requirements
  - NOT a meaningless ritual; document not subject to arbitrary change
  - IS a baseline from which the impacts of changes can be assessed, especially in time, $$, and resources
Good Practices for Requirements Engineering

- [http://www.processimpact.com/articles/customer.html](http://www.processimpact.com/articles/customer.html) (go now if time!)
- Apply selectively and appropriately
- A *lifecycle model* provides a framework for understanding the appropriateness and impact of the practice
- Types of Practices
  - Knowledge
  - Requirements management
  - Project management
  - Requirements development
    - Elicitation
    - Analysis
    - Specification
    - Verification
Project Vision and Scope: Milestone 1
(Figure 5-2)

1. Business requirements
   – Background
   – Business opportunity
   – Business objectives
   – Customer or market requirements
   – Value provided to customers
   – Business risks

2. Vision of solution
   – Vision statement
   – Major features
   – Assumptions and dependencies
Project Vision and Scope:

3. Scope and limitations
   - Scope of initial release
   - Scope of subsequent releases
   - Limitations and exclusions

4. Business context
   - Stakeholder profiles
   - Project priorities
   - Operating environment

5. Product success factors
   - How will success be defined, measurable criteria
The Context Diagram

• Graphical illustration of the system and how it relates to the outside world
  – users
  – other application software
  – databases
  – defines the “boundaries” of the system to be built

• Can be part of the Vision and Scope document but also in the Software Requirements Specification
What about Reality?

- Wiegers recognizes Jackson’s ideas about context diagrams
• The top level of a hierarchical collection of dataflow diagrams
  – process in the middle, the system to be developed
  – rectangles represent sources or sinks for system data
• The main focus of the diagram is the system to be developed
  – not the sources or sinks (DeMarco back in 1978)
  – the “context” is for the system, the machine, not the “problem”
• Jackson thinks this should be more of a problem context diagram
  – show all the domains that are relevant to the problem, not just direct sources or sinks for the machine
  – loose notion of connections between domains (not just dataflow)
  – the machine element does not have a special symbol
Simple Problem

Patient Monitoring System

A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temp, bp, and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient’s safe range, or if an analog device fails, the nurses’ station is notified.
“Context”

• Every context diagram has exactly one machine domain
• All domains in the context diagram are physical (not conceptual)
  – parts of the world where your customer will check for observable effects
• The **machine domain** is the computer (we design and build by creating its software)
• A **designed domain** is the physical representation of some information
  – that we are free to design to some extent
    • such as on a magnetic strip card or HD.
• A **given domain** is a problem domain whose properties are given
  – we are not free to design the domain
    • such things must be a part of our analysis but not design
Shared Phenomena

- The domains are physical
  - the interfaces between them are also physical
  - don’t think of them as streams or pipes
- THINK: events, states, values that are shared between connected domains
- Each interface is an interface of shared phenomena
  - In our example, the interface between machine and the nurses’ station consists of shared “notify” events. The machine can cause a notify event, and both the machine and the nurses’ station then participate in this same event
  - Example - an ICU patient has a certain skin resistance: that is a state shared with an attached analog device.
  - There is NO notion of patients sending messages to analog devices or analog devices sending messages to the machine.
    » both of the sharing participants can see the state or value, but only one of them can change it…
Problem Context diagram

- Patient monitoring system
• Patients domain included even though no direct connection
  – patients are of central interest in the problem
• Analog devices domain is included as integrity of data must be checked
• This diagram is more abstract than DeMarco’s
  – and should be provided as the higher level view
  – the other view should also be provided if the information is meaningful
Context: Boundary of the Problem to be Solved

- Circular relation between problem and its context
  - iterative process between problem and context knowledge
Project Risks (at the Requirements level of development)

• **No** risk management is potentially costly to a project
  – as is the lack of configuration control, defect tracking, productivity or schedule
  – lots of data to show this is a serious ongoing problem
  – always remember “costs and benefits” of any process
    • the right balance to match individual project factors such as size, dollar amount, other QA factors
    • in this class we only consider larger projects of some complexity where benefits or risk management have proven necessary

• **Risk** is a condition that could cause some loss or otherwise threaten the success of a project
  – hasn’t arisen yet, but we’d like to keep it from arising or doing too much damage (or get out before it does :-)

January 16, 2009
Fundamentals of Risk Management

- Some common risks
  - scope and requirements creep
  - dependencies on external entities
    - subcontractor
    - other COTS expected to be used
- Common causes of risks
  - poor estimation
  - rejection of accurate estimates
  - insufficient visibility into project status
  - staff turnover
  - micro management in the way of the work
Elements of Risk Management

• Risk management consists of the following:
  – Risk assessment
    • identification (of potential risks)
    • analysis (potential consequences)
    • prioritization (probability times consequence potential gives risk exposure)
  – Risk avoidance (don’t do the risky thing!)
  – Risk control
    • management planning
      – mitigation, contingency plans, owners of risk items, timelines
    • resolution
      – executing plans for mitigating / resolving each risk
    • monitoring
      – how well the plans are working, review the plans given current state of process
Documenting Risks in a Project

• Use a *condition-consequence* format to document risk statements
  – one condition may have several possible consequences
  – several conditions may contribute to the same consequences
  – entire disciplines built around analytical tools to deal with project risks
    • fault tree analysis
    • failure modes and effects analysis
    • lots more
• Use a Risk Item Tracking form
  – use common sense
    • don’t spend 20 hours on an item of very small risk potential
    • don’t forge ahead if the entire project depends on a very risky item
  – you will **track the top three risks** for your requirements project
  – follow them up and keep them current
Requirements related risks list

- Requirements elicitation
  - scope creep
    - project vision and scope should help avoid
  - time spent on this stage
  - completeness, correctness
    - can write usage scenarios, test cases, prototypes
  - highly innovative projects necessarily involve risk
  - nonfunctional requirements notoriously difficult
    - usability, reliability, safety, speed...
  - customer agreement on product requirements
    - takes two consenting adults to agree
  - unstated requirements and assumptions
  - reverse engineering is notoriously difficult
- solutions presented as needs
  - precludes a lot of design flexibility

- Requirements Specification
  - gaps in specifier / customer understanding
    - formal inspections including all stakeholders have significant impacts on this
  - time pressure
    - leaving important TBD’s unresolved can be destructive
      - assign responsibility for TBD’s and enforce accountability
  - vocabulary problems
    - creates misunderstandings
      - early creation (maintenance) of data dictionaries and glossaries
  - requirements that are actually design (distinctions?)
    - unnecessary constraints on the designers
• Requirements verification
  – use formal inspection, test planning, write user manuals
    • recall the costs of fixing “problems” later in the lifecycle
    • requires commitment and followthrough from customers / users
      – informal, quick pre-reviews may be helpful at the outset
    • requires training of ALL members of verification teams in relevant methods
      – include experienced people

• Requirements management
  – dynamism
    • scope creep happens
      – risk work
        » delay implementation of changeable requirements till thoroughly understood
        » design for modifiability
– change process
  • must be carefully defined and respected!
  • supported by a culture of respect for the process
  • impact analysis, change control board to make decisions, tool to implement
– traceability / forgotten requirement
  • traceability matrix
    – responsibility and follow up critical
– scope expansion
  • incremental or phased delivery to iteratively elaborate requirements
Who is the Customer? Where do I get User Requirements?

Basic steps:
- identify *sources of user requirements*
- identify *classes of users* for the project
- gain access to *individuals who represent the user classes*
- agree who is the *ultimate decisionmaker* for the project

Become/remain aware of who are the
- enemies of the project
- losers if the project is successful

**WHY** is the project really being undertaken?
- this affects everything!
Sources of Requirements

- Meet the potential users
- Market research in the domain
  - other products
  - market surveys
- System requirements specifications
  - get product spec sheets when possible (marketing materials)
- Change requests and bug reports from a current system
- Observation or users
- Scenario analysis
  - developed into the use-case approach
User Classes

• Different tasks may be required if userbase not homogeneous
  – big task if you are working on a meta application to generalize for all possible user classes
• Simple example: physician’s office automation may need to serve -
  – M.D.’s
  – secretaries
  – nurses
  – physician’s assistants
  – insurance companies (via machine interfaces)
  – laboratory technicians
  – DEA auditors
• Find classes, characterize them and document in the SRS
Responsibility - Find a Representative for the User Classes

- User centered development: users should be involved throughout the lifecycle
  - investment of time and energy towards the goal of higher quality products (products that more effectively meet the user’s needs)
  - users who represent user classes must be chosen carefully
    - product “champion” approach
  - pay attention to risks you assume by choosing user representatives
    - like the marketing department :-)

January 16, 2009
Product Champion Approach

- Key participant in development
  - accurate perspective on user class: an actual user
    - who cares about the project
    - in regular communication with other users
    - who is supported by their management (time, money)
    - experience with the problem domain (and technology) is important
  - collects requirements from the class
    - the champion must have standing in the user community
  - responsible for decisionmaking when difficulties arise
    - for best results managers must respect the champion’s decisions in most cases
  - developers might want to pay them if critical to the project!
    - or hire a champion separately as part of the development team
  - team up with analyst to write user requirements for user class