What are Software Requirements?

Condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document

- Description of what the customers wants and what the system has to do
- Functionality of the features, that the system has to deliver in order to satisfy its stakeholders
- May be buried beneath layers of assumptions, misconceptions, and politics

Q: Who are the stakeholders for your Sen Design project?
## WHY PROJECTS FAIL

<table>
<thead>
<tr>
<th>Reason</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete Requirements</td>
<td>13.1</td>
</tr>
<tr>
<td>Lack of user involvement</td>
<td>12.4</td>
</tr>
<tr>
<td>Lack of resources</td>
<td>10.6</td>
</tr>
<tr>
<td>Unrealistic Expectations</td>
<td>9.9</td>
</tr>
<tr>
<td>Lack of executive support</td>
<td>9.3</td>
</tr>
<tr>
<td>Changing requirements</td>
<td>8.7</td>
</tr>
<tr>
<td>Lack of planning</td>
<td>8.1</td>
</tr>
<tr>
<td>Didn’t need it any longer</td>
<td>7.5</td>
</tr>
<tr>
<td>Lack of IT management</td>
<td>6.2</td>
</tr>
<tr>
<td>Technology illiteracy</td>
<td>4.3</td>
</tr>
</tbody>
</table>

### Relative Cost to Fix an Error

<table>
<thead>
<tr>
<th>Phase in Which Found</th>
<th>Cost Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1</td>
</tr>
<tr>
<td>Design</td>
<td>3-6</td>
</tr>
<tr>
<td>Coding</td>
<td>10</td>
</tr>
<tr>
<td>Development testing</td>
<td>15-40</td>
</tr>
<tr>
<td>Acceptance testing</td>
<td>30-70</td>
</tr>
<tr>
<td>Operation</td>
<td>40-1000</td>
</tr>
</tbody>
</table>

Boehm's analysis of 63 s/w development projects (IBM, GTE, TRW, etc.) to determine ranges in cost for errors created by false assumptions in req'ts phase but not detected till later phases.
Requirements Engineering

Systematic way of developing requirements through an iterative process

- to **elicit and proactively discover** the requirements
- **analyze** requirements to understand tradeoffs and figure out feasibility
- **constraints** under which the system operates and is developed
- **resolve conflicts**
- **validation**
- modeling, translating requirements the user understands to a form that software engineers understand
- **requirements management**

→ results in a **Software Requirements Specification**
Completeness and Pertinence

Difficult to identify ALL the requirements - missing functionality for the user

Irrelevant Requirements
Pertinence Example

Consider an information system for a gym. Which of the listed requirements are pertinent?

A. Members of the gym shall be able to see the training schedule
B. The system shall be able to read members’ cards
C. The system shall be able to store member’s commute time
D. Personal trainers shall be able to add clients
E. The list of members shall be stored as a linked list
Types of Requirements

<table>
<thead>
<tr>
<th>Functional</th>
<th>Non-Functional</th>
<th>Constraints</th>
</tr>
</thead>
</table>
| Functional requirements specify a function that a system or system component must be able to perform | Non-functional requirements are not specifically concerned with the functionality of a system but place restrictions on the product being developed. They include:  
- User visible aspects of the system not directly related to functional behavior  
- Usability, reliability, privacy, security, availability, performance,  
- Best to translate non-functional to measurable. | Constraints are ("Pseudo requirements"): not user-visible; imposed by the client that restricts the implementation of the system or the development process. |

- The watch shall display the time.  
- The response time must be less than 1 second  
- The implementation language must be Java.
Requirements Elicitation

• Need to understand **WHY** not just **WHAT**.

• Techniques:
  – Interviews
  – Observation
  – Examining Documents and Artifacts
  – Join Application Design Sessions (JAD)
  – Groupware
  – Questionnaires
  – Prototypes
  – Focus Groups
  – On-Site Customer

In Senior Design:
• Proposal
• Kick-Off Meeting
• Weekly Sponsor Meetings
• Various Supporting Documents provided by sponsors
Requirements Validation

Critical step in the development process, usually after requirements engineering or requirements analysis.

- Correctness:
  - The requirements represent the client’s view.
- Completeness:
  - All possible scenarios through the system are described, including exceptional behavior by the user or the system
- Consistency:
  - There are no functional or nonfunctional requirements that contradict each other
- Clarity:
  - There are no ambiguities in the requirements.
- Concise
  - Don’t “boil the ocean” or extend beyond what is in current release

- Feasible:
  - Requirements can be implemented and delivered
- Traceability:
  - Each system function can be traced to a corresponding set of functional requirements
- Understandable
- Non-prescriptive
  - everything about what the customer wants and nothing about how the programmer(s) will do it.
- Consistent language
  - Shall, should, may
  - “the physician” vs. “the doctor”
- Testable
Software Requirements Specification (SRS)

System Requirements should be

- simple (non compound)
- testable
- organized
- numbered (for traceability)
Types of Requirements Statements

• **Traditional** - list of “shall”- type statements

• **Use Cases** - describe how the system will act
  – you have seen these in CSC326 in iTrust

• **User Stories** – captures what a user does or needs to do as part of her work
  – with user role, goal and acceptance criteria
  – centered on the result and the benefit of the thing you're describing
Traditional Requirement Statements

- **Shall** (**is required to**): used to indicate mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted.

- **Should** (**is recommended that**): used to indicate
  - among several possibilities one is recommended as particularly suitable, without mentioning or excluding others
  - certain course of action is preferred but not necessarily required;
  - certain course of action is deprecated but not prohibited (in the negative form).

- **May** (**is permitted to**): used to indicate a course of action permissible within the limits of the standard.

- **Can** (**is able to**): used for statements of possibility and capability, whether material, physical, or causal.

See Section 11.2.2

https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf
Traditional Requirements

FUNCTIONAL REQUIREMENTS

FR2.4 Go to Jail
When the player lands on the Go to Jail cell, the player shall be sent to the Jail cell. The player shall not receive $200 if she or he passes the Go cell on the way to the Jail cell.

FR2.5 Buy Property
When the player lands on a tradable cell, including properties, railroads, and utilities, she or he shall have a chance to buy that cell given that the cell is available.

NON-FUNCTIONAL REQUIREMENTS

NFR1.1 User Response
The system shall respond to any user input within 0.01 seconds.

CONSTRAINTS
• All code development shall be done with the Java programming language
• All testing shall be done using JUnit and FIT
Use Case-Based Requirement Elicitation

• Use cases describe functional aspects of the system

• UML focuses on **scenario-based** requirements elicitation

• Scenario:
  – sequence of actions that illustrates behavior.
  – A scenario may be used to illustrate an interaction or the execution of a use case instance
Use Case

• A sequence of transactions performed by a system that yields a measurable result of values for a particular actor

• A use case typically represents a major piece of functionality that is complete from beginning to end. A use case must deliver something of value to an actor.

• Use cases that an actor “wants” begin with verbs.
Template for **Flow of Events**

X Flow of Events for the `<name>` Use Case

X.1 Preconditions

What needs to happen (in another use case) before this use case can start?

X.2 Main Flow

X.3 Subflows

Break “normal” flow into pieces

“called” by Main Flow or another subflow

X.4 Alternative Flows

Things that happen outside of the “normal” flow

“called” by Main Flow or a subflow

➔ Covers multiple related scenarios!!!
Requirements in Senior Design

Written in a **Traditional Format** – “The system shall…."

- Requirements must be numbered
- Related requirements should be grouped under subheadings
- Each functional traditional requirement should have an associated acceptance test!

Requirements may also be written as **Use Cases**

- Use cases are an excellent tool to identify related scenarios and can drive creation of traditional requirements
- Sponsors like to discuss scenarios

**Combination**

- You can model requirements as traditional requirement statements, use cases, and/or both.
- Iteration between traditional statements and use cases can be useful for fully understanding the system.
Non-Functional Requirements Example

What do you think about these? How can they be improved?

NFR 1. The system should not negatively impact the performance
NFR 2. User friendly
NFR 3. Aesthetically pleasing
Requirements Gathering Example

Suppose that you were tasked to write a program that determines the average length of sentences in an essay. What is a relevant question to ask?

1. What OS(es) should it run on?
2. How will the user specify input?
3. How many lines of code should it take?
References

• CSC326 and CSC492 NCSU Faculty

• Software Development Process Course -- Udacity, Alessandro (Alex) Orso is a Professor in the College of Computing at Georgia Tech.