CSCI 4050/6050
Software Engineering

Requirements Elicitation and Specification

Requirements Types

Dr. Eman Saleh eman.saleh@uga.edu

What is a Requirement?

A **requirement** is a statement of one of the following:

1. What a system must do
2. A known limitation or constraint on resources or design
3. How well the system must do what it does

The first definition is for **Functional Requirements**

The second and third definitions are for **Non-Functional Requirements**

Types of Requirements

- Functional requirements
  - Describe the interactions between the system and its environment independent from the implementation
    "An operator must be able to define a new game. "
- Nonfunctional requirements
  - Aspects not directly related to functional behavior.
    "The response time must be less than 1 second"
- Constraints
  - Imposed by the client or the environment
    - "The implementation language must be Java"
    - "The persistence module must use POSTGRES RDBMS"
  - Also called "Pseudo requirements"

Functional vs. Nonfunctional Requirements

Functional Requirements

- Describe user tasks that the system needs to support
- Phrased as actions
  - "Advertise a new league"
  - "Schedule tournament"
  - "Notify an interest group"
  - "Transfer funds from one account to another"
  - "Post an item for sale"
  - "Place a bid for an item"

Nonfunctional Requirements

Describe properties of the system or the domain

Phrased as constraints or negative assertions

- "All user inputs should be acknowledged within 1 second"
- "A system crash should not result in data loss".

Types of Nonfunctional Requirements

Quality requirements

Constraints or
Pseudo requirements

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
  - Security
- Performance
  - Response time
  - Throughput
- Supportability
  - Adaptability
  - Maintainability
  - modifiability

Quality requirements

Constraints or
Pseudo requirements

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
  - Security
- Performance
  - Response time
  - Throughput
- Supportability
  - Adaptability
  - Maintainability
  - modifiability

Quality requirements

- Implementation (must be in C++)
- Interface (must be Web-based)
- Operation
- Packaging
- Legal
  - Licensing (GPL, LGPL)
  - Certification
  - Regulation

Constraints or
Pseudo requirements

Review at Home

- Find definitions for all the nonfunctional requirements on the previous slide and memorize them
- Understand their meaning and scope (their applicability).

Some Quality Requirements Definitions

- Usability
  - The ease with which actors can use a system to perform a function
  - Usability is one of the most frequently misused terms ((“The system is easy to use”)
  - **Usability** must be **measurable**, otherwise it is **marketing**
    - Example: Specification of the number of steps – the measure! - to perform a internet-based purchase with a web browser
- Robustness: The ability of a system to maintain a function
  - even if the user enters a wrong input
  - even if there are changes in the environment
    - Example: The system can tolerate temperatures up to 90 C
- Availability: The ratio of the expected uptime of a system to the aggregate of the expected up and down time
  - Example: The system is down not more than 5 minutes per week.

Product-oriented attributes (1)

- Performance : (a) response time, (b) throughput (number of operations performed per second)
- Usability: effort required to learn, use, provide input and interpret results of a program
- Efficiency: minimal use of resources (memory, processor, disk, network…)
- Reliability: of computations, precision

Product-oriented attributes(2)

- Security
- Robustness: in the presence of faults, stress, invalid inputs…
- Adaptability: to other environments or problems
- Scalability: for large number of users or quantities of data
- Cost: total cost of ownership (TCO) for acquisition, installation, use,

Product family-oriented attributes

- Portability: does it work for several platforms
- Modifiability: addition of new functionalities
- Reusability: of components, code, designs, and even requirements in other systems

Process-oriented attributes

- Maintainability: changes to functionalities, repairs
- Readability: of code, documents
- Testability: ease of testing and error reporting
- Understandability: of design, architecture, code
- Integrability: ability to integrate components

Nonfunctional Requirements: Examples

- “Spectators must be able to watch a match without prior registration and without prior knowledge of the match.”
- “The system must support 10 parallel tournaments”
- “The operator must be able to add new games without modifications to the existing system.”

Nonfunctional Requirements: Examples

- “Spectators must be able to watch a match without prior registration and without prior knowledge of the match.”
  - *Usability Requirement*
- “The system must support 10 parallel tournaments”
- “The operator must be able to add new games without modifications to the existing system.”

Nonfunctional Requirements: Examples

- “Spectators must be able to watch a match without prior registration and without prior knowledge of the match.”
  - *Usability Requirement*
- “The system must support 10 parallel tournaments”
  - *Performance Requirement*
- “The operator must be able to add new games without modifications to the existing system.”

Nonfunctional Requirements: Examples

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."
  - *Usability Requirement*
- "The system must support 10 parallel tournaments"
  - *Performance Requirement*
- "The operator must be able to add new games without modifications to the existing system."
  - *Supportability Requirement*

What should not be in the Requirements?

- System structure, implementation technology
- Development methodology
- Development environment
- Implementation language
- Reusability

- **It is desirable that none of these above are constrained by the client.**

Requirements Validation

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis

- Correctness:
  - The requirements represent the client's view
- Completeness:
  - All possible scenarios (or user stories), in which the system can be used, are described
- Consistency:
  - There are no requirements that contradict each other.

Requirements Validation (2)

- Clarity:
  - Requirements can only be interpreted in one way
- Realism:
  - Requirements can be implemented and delivered
- Traceability:
  - Each system behavior can be traced to a set of functional requirements
- Verifiability : Should be written so that they could be tested. Means that we should be able to write a set of tests that can demonstrate that the delivered system meets its specified requirements.

Requirements Validation (2)

- Problems with requirements validation:
  - Requirements change quickly during requirements elicitation
  - Inconsistencies are easily added with each change
  - Tool support is needed!

To measure is to know. If you can not measure it, you can not improve it.

[Lord Kelvin (1824 - 1907)]

Verifiable Non-Functional Requirements

- We need to be able to explicitly quantify requirements and verify that any solution meets them

- We need measures

Avoid subjective characterization: good, optimal, better, easy to use...

Verifiable Non-Functional Requirements

- The chosen values for the measues will have an impact on the amount of work during development as well as the number of alternatives and architectural designs from which developers may choose to meet the requirements

Performance Measures

- Lots of measures:

Response time, number of events processed/denied in some interval of time, throughput, capacity, usage ratio, jitter, loss of information, latency...

- Usually with probabilities or confidence interval

- Examples of performance requirement:
- The system shall be able to process 100 payment transactions per second in peak load.

Reliability Measures (1)

- Measure degree to which the system performs as required
  - Includes resistance to failure
  - Ability to perform a required function under stated conditions for a specified period of time
  - Very important for critical, continuous, or scientific systems

Reliability Measures (2)

- Can be measured using
  - Probability that system will perform its required function for a specified interval under stated conditions
    - Mean-time to failure
    - Defect rate
    - Degree of precision for computations

Example

- The precision of calculations shall be at least $1/10^6$.

Availability Measures

- Definition: Percentage of time that the system is up and running correctly
- Can be calculated based on Mean-Time to Failure (MTBF) and Mean-Time to Repair (MTTR)
  - MTBF : Length of time between failures
  - MTTR : Length of time needed to resume operation after a failure

Availability = MTBF/(MTBF+MTTR)

- May lead to architectural requirements
  - Redundant components (lower MTBF)
  - Modifiability of components (lower MTTR)
  - Special types of components (e.g., self-diagnostic)

Availability Measures : Example

- The system shall meet or exceed 99.99% uptime.

Availability	Downtime
90%	36.5 days/year
99%	3.65 days/year
99.9%	8.76 hours/year
99.99%	52 minutes/year
99.999%	5 minutes/year
99.9999%	31 seconds/year

Security Measures

- There are at least two measures:

  1. The ability to resist unauthorized attempts at usage

  2. Continue providing service to legitimate users while under denial of service attack (resistance to DoS attacks)

Security: Measurement Methods

- Success rate in authentication
- Resistance to known attacks (to be enumerated)
- Time/efforts/resources needed to find a key (probability of finding the key)
- Probability/time/resources to detect an attack
- Percentage of useful services still available during an attack
- Percentage of successful attacks
- Lifespan of a password, of a session
- Encryption level

Security Requirements: examples

- The application shall identify all of its client applications before allowing them to use its capabilities.

Nonfunctional Requirements as user stories

- Constraints/non-functional requirements can be easily handled as user stories. Here are a couple of examples:
- As a customer, I want to be able to run your product on all versions of Windows from Windows 95 on.
- As the CTO, I want the system to use our existing orders database rather than create a new one, so that we don't have one more database to maintain.
- As a user, I want the site to be available 99.999 percent of the time I try to access it, so that I don't get frustrated and find another site to use.
- As a user, I want the driving directions to be the best 90 percent of the time, and reasonable 99 percent of the time.

Examples from: https://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories

We can specify Requirements for "Requirements Management"

- Functional requirements:
  - Store the requirements in a shared repository (user stories are often pinned on a large board)
  - Provide multi-user access to the requirements
  - Automatically create a specification document from the requirements
  - Allow change management of the requirements
  - Provide traceability of the requirements throughout the artifacts of the system.

Nonfunctional Requirements (Questions to overcome "Writers block")

User interface and human factors

- What type of user will be using the system?
- Will more than one type of user be using the system?
- What training will be required for each type of user?
- Is it important that the system is easy to learn?
- Should users be protected from making errors?
- What input/output devices are available

Documentation

- What kind of documentation is required?
- What audience is to be addressed by each document?

Nonfunctional Requirements (2)

Hardware considerations

- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Performance characteristics

- Are there speed, throughput, response time constraints on the system?
- Are there size or capacity constraints on the data to be processed by the system?

Error handling and extreme conditions

- How should the system respond to input errors?
- How should the system respond to extreme conditions?

Nonfunctional Requirements (3)

System interfacing

- Is input coming from systems outside the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?

Quality issues

- What are the requirements for reliability?
- Must the system catch faults?
- What is the time for restarting the system after a failure?
- Is there an acceptable downtime per 24-hour period?
- Is it important that the system be portable?

Nonfunctional Requirements (4)

System Modifications

- What parts of the system are likely to be modified?
- What sorts of modifications are expected?

Physical Environment

- Where will the target equipment operate?
- Is the target equipment in one or several locations?
- Will the environmental conditions be ordinary?

Security Issues

- Must access to data or the system be controlled?
- Is physical security an issue?

Nonfunctional Requirements (5)

Resources and Management Issues

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?

Prioritizing requirements

- High priority
  - Addressed during *analysis, design, and implementation*
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during *analysis and design*
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed *only during analysis*
  - Illustrates how the system is going to be used in the future with not yet available technology

Different Types of Requirements Elicitation

- Greenfield Engineering
  - Development starts from scratch, no prior system exists, requirements come from end users and clients
  - Triggered by user needs
- Re-engineering
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
- Interface Engineering
  - Provision of existing services in a new environment
  - Redesign of the user interface (UI) of an existing system, The legacy system is untouched except for its the UI
  - Triggered by technology enabler or new market needs

Tools for Requirements Management

DOORS (IBM/Telelogic)

- Multi-platform requirements management tool, for teams working in the same geographical location. DOORS XT for distributed teams

RequisitePro (IBM/Rational)

- Integration with MS Word
- Project-to-project comparisons via XML baselines

Atlassian JIRA (Atlassian)

- Can be used to manage agile projects and track bugs.

OpenProject (OpenProject Foundation)

- Open source project collaboration sotware; GNU license
- Timelines, backlogs, wikis
- Includes SCRUM plugin

Additional Readings

- *Scenario-Based Design: Envisioning Work and Technology in System Development*, by John M. Carrol, John Wiley, 1995.
- *Requirements Analysis and System Design (3rd Ed.)*, by Leszek A. Maciaszek, Addison Wesley, 2007.
- *Use Case Driven Object Modeling with UML. Theory and Practice*, by Doug Rosenberg and Matt Stephens, Apress, 2007.
- *Use Cases: Patterns and Blueprints*, by Gunnar Overgaard and Karin Palmkvist, Addison-Wesley, 2004.
- *Writing Effective Use Cases*, by Alistair Cockburn, Addison-Wesley, 2000.
- *Advanced Use Case Modeling: Software Systems*, by Frank Armour, Addison-Wesley, 2000.
- *Use Cases: Requirements in Context (2nd Ed.)*, by Daryl Kulak and Eamonn Guiney, Addison-Wesley, 2003.