CS60.311- Fall 2018

University of Windsor

Introduction to Software Engineering

Lecture 03.2: Requirements Engineering Part 2

Agenda

- Requirements Prioritization
- Requirements Specification
- Requirements Validation

Requirements Prioritization

Requirements Prioritization

What is Software Requirements Prioritization (SRP)?

- The process of analyzing software requirements to identify important requirements.
- Determine the essential functionalities that should be implemented as early as possible.
- Determine which non-functional requirement to be implemented to obtains customers satisfaction.

Requirements Prioritization (cont.)

Who is responsible for the Requirements Prioritization?

- The stakeholders should prioritize the requirements. This includes the end-users, developers, project managers, customers, and business managers.
- Note that not all the stakeholders have the same importance (e.g., weight).
- The primary users are usually the most important class of stakeholders, but again different types of primary users are not equal.

Requirements Prioritization (cont.)

Why Requirements Prioritization is important?

- The success of a software system is often determined by the ability to satisfy the requirements of the stakeholders.
- Software systems that are developed based on prioritized requirements usually have a lower risk of being rejected.
- The number of users' requirements that can be implemented within the given time and available resources is usually a subset of all the requirements.
- To trade off desired project scope against sometimes conflicting constraints such as schedule, budget, resources, time to market, and quality.

Requirements Prioritization Techniques

There are many requirements prioritization techniques.

1. Nominal Scale Prioritization Techniques.
2. Ratio Scale Prioritization Techniques.
3. Dependency Prioritization Techniques.

Nominal Scale Prioritization Techniques

- Also known as numerical assignment techniques.
- Requirements are categorized into groups based on their importance.
- All requirements in one priority group have equal priority.
- Used in the early stages of project when requirements are not specified in a great level of details.
- Better when prioritizing medium or large number of requirements.
- The most common technique for prioritization of requirements
- **Examples:** Top-K requirements, MoSCoW technique [MUST, SHOULD, COULD, WHON'T].

Top-Ten (K) Requirements

- The stakeholders pick their top-ten requirements (from a larger set) without assigning an internal order between the requirements.
- The stakeholders do not rank the K requirements.
- The development team identifies the requirements that will be implemented in the system based on the most common or shared requirements between the stakeholders.
- The requirements are categorized into groups (e.g. critical, standard, and optional)
- Suitable for multiple stakeholders of equal importance.

Ratio Scale Prioritization Techniques

- Produce ranked lists of requirements.
- Techniques in this category show the relative difference between requirements.
- Provide an answer to ”How much important is this one requirement when compared to another?”
- More sophisticated and complex comparing to the nominal scale prioritization techniques.
- **Examples:** Analytic Hierarchy Process (AHP) and 100-Dollar Test, Ranking (Binary Tree, Bubble sort)

Analytic Hierarchy Process (AHP)

- A systematic decision-making method developed in the 1980s and adapted for prioritization of software requirements by Karlsson in mid 1990s
- Uses pair-wise comparisons, that means for n number of requirements there are n (n−1)/ 2 comparisons.
- Calculate the priorities of requirements by comparing all unique pairs of requirements to estimate their relative importance.
- Suitable for prioritizing small number of requirements

100-Dollar Test (Cumulative Voting)

- Simple and straightforward prioritization technique.
- Every stakeholder is given 100 imaginary units (money, point, etc.) to distribute between the requirements.
- The number of units assigned to a requirement represents its priority.
- The result of the prioritization is presented on a ratio scale which provides the information on how much one requirement is more/less important than another.
- Less complex than AHP and more reliable than MoSCow technique.

Binary Priority List (BPL)

- In this technique the requirements are prioritized using a binary search tree. From a set of all requirements take any one requirement and put it as the root requirement (root of the tree)
- Take another requirement and compare it to the root requirement.
- If the requirement has a lower priority than the root requirement, compare it to the requirement to the left the root requirement. If the requirement has higher priority than the root requirement, compare it to the requirement to the right the root requirement.
- Repeated the previous 2 steps for all requirements.
- Finally, traverse the tree using **in-order** traversal method.

Dependency Prioritization Techniques

- This technique takes into account the dependency between the requirements when prioritizing.
- In addition to the value of the requirement to the stakeholder, we will consider the dependency between the requirements when we are ranking them.
- This means even if a given requirement SR5 has a higher value than SR2 based on the system user, but to implement SR5, we must first implement SR2 then SR2 will likely be ranked higher than SR5
- **Examples:** Dependency map and Dependency graph

Requirement Prioritization Example 1

In an online exam system, we have four different users, namely, System Admin, Instructor, Proctor, and Student. In total, there are 15 requirements (SR01 - SR15). Each user uses the **100-Dollar test** to indicate the importance/priority of the requirements. Given priority assigned by each user and the weights of the user are **Admin (0.15)**, **Instructor (0.30)**, **Proctor(0.20)**, and **Student(0.35)**. Calculates the overall priority of the requirements.

Requirement Prioritization Example 1

$$Priority(R) = \sum_{i}^{u} (W_i \times RP_i) \tag{3}$$

Req	**Admin (0.15)**	**Instructor(0.30)**	**Proctor(0.20)**	**Student (0.35)**	**Priority**
R8	0.25	0.24	0.16	0.15	0.19
R9		0.07	0.14	0.03	0.06
R10	0.25	0.05	0.13	0.29	0.18
R11		0.05	0.01	0.02	0.02
R12		0.16	0.04	0.01	0.06
R13		0.05	0.16	0.02	0.05
R14	0.25	0.02	0.10	0.10	0.10
R15		0.03	0.04	0.05	0.03
R3,4,5		0.04	0.18	0.17	0.11
R6,7	0.25	0.29	0.04	0.16	0.19
Total	**1**	**1**	**1**	**1**	**1**

Requirement Prioritization Example 2

In the online exam system from the previous example we have four important non-functional requirements. These requirements are **availability**, **security**, **performance**, and **usability**. Rank these non functional requirements using a point system

Requirement Prioritization Example 2

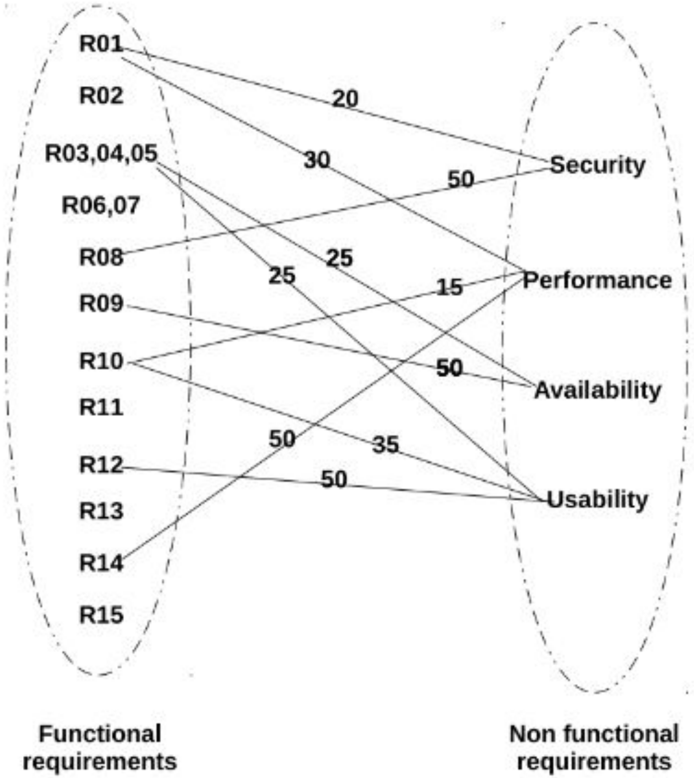
$$Priority(NFR) = \sum_{i}^{R} P_i \times log_2 \left(\sum weights(NFR_i) \right)^{degree(NFR_i)}$$

Priority(Availability) = $0.17 \times \log_2(75)^2$

Priority(Usability) = $0.35 \times \log_2(110)^3$

Priority(Security) = ???

Priority(Performance) = ???



Requirement Prioritization Example 3

Give the 8 requirements X, Y, Z, P, Q, R, M, O and N with priorities, on a 5- level scale where 1 is most critical and 5 least critical, as 1, 2, 1, 4, 5, 1, 2, 2, 3. The dependency between the 8 requirements is **[Y, Q] depend on M** and **[Q, X, P] depend on R and N**. Rank the 8 requirements using dependency graph.

Requirement Prioritization Example 3

Solution given in the class

Requirements Specification

Requirement Specification

- The process of **writing down the user and system requirements** in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development, therefore it is important to be consistent and complete as possible

Documenting System Requirements Specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don’t understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Insulin Pump System: A Running Example

A personal insulin pump is an external device that mimics the function of the pancreas It uses an embedded sensor to measure the blood sugar level at periodic intervals and then injects insulin to maintain the blood sugar at a ‘normal’ level.

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.
- The most common method to write requirements, in particular for custom software projects.
- Precision is difficult without making the document difficult to read.
- Functional and non-functional requirements tend to be mixed-up.

Insulin Pump Requirements in Natural language

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured Specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements, e.g., requirements for embedded control system but is sometimes too rigid for writing business system requirements.
- In general, the specification author is limited to specific keywords and vocabularies based on the system domain.
- Using form-based specification is a common method to write structured specifications.

Form-based Specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Insulin Pump Requirements Form-based Specification

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

Insulin Pump Requirements Form-based Specification

Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Tabular Specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Insulin Pump Requirements: Tabular Specification

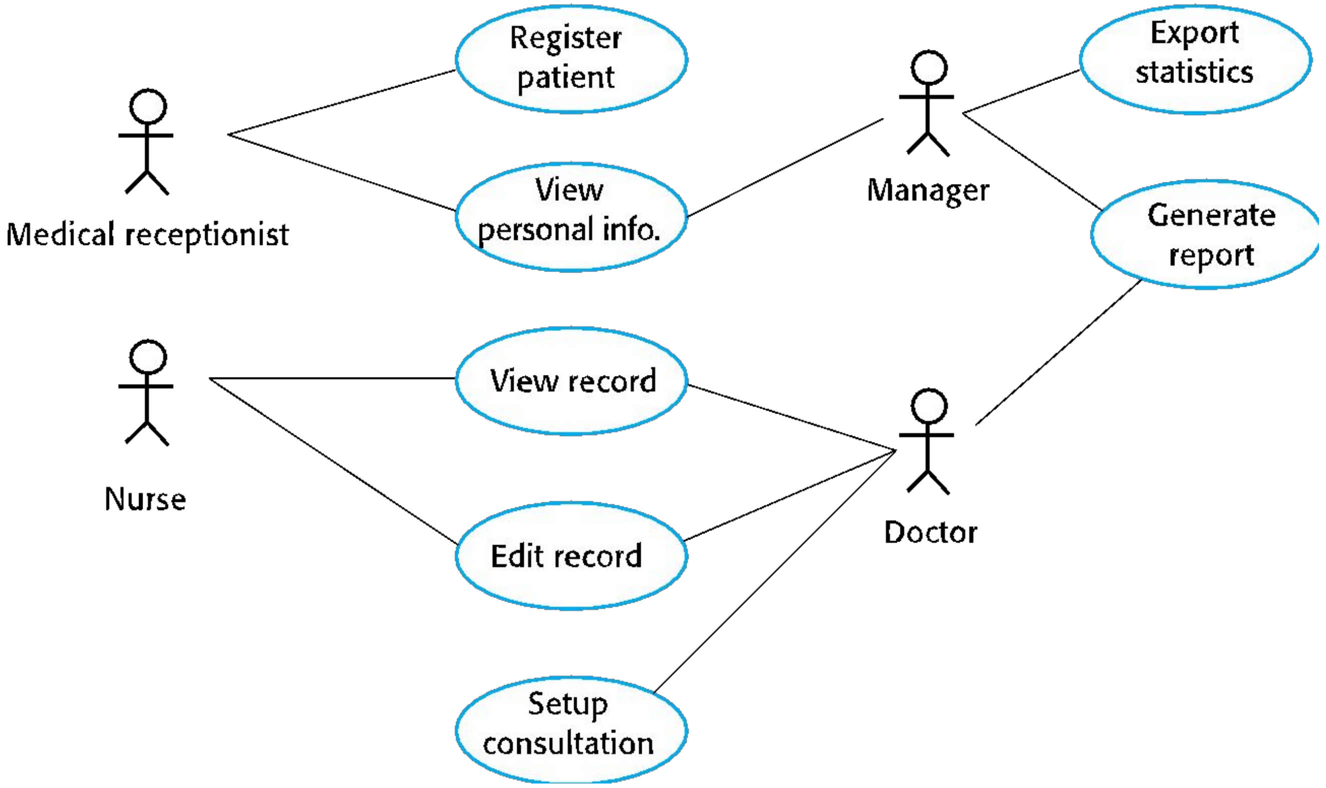
Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing $((r2 – r1) < (r1 – r0))$	CompDose = 0

Sugar level increasing and rate of increase stable or increasing $((r2 – r1) \geq (r1 – r0))$	CompDose = round $((r2 – r1)/4)$
	If rounded result = 0 then
	CompDose = MinimumDose

Use Cases

- Use-cases are a kind of scenarios that are included in the UML.
- Use cases identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description.
- UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

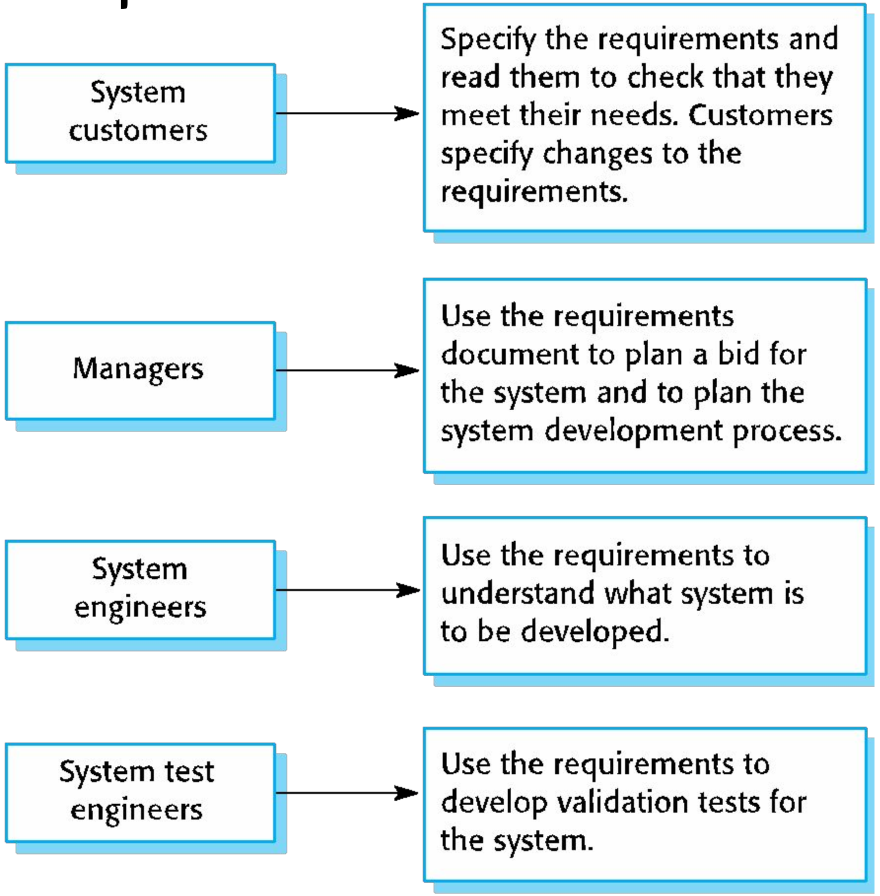
Use Cases in Mencare System



Software Requirement Document

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Users of the specification document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.

Requirements Document Variability

Information in requirements document depends on the type of system and the approach to development used.

Systems developed incrementally will, typically, have less detail in the requirements document.

Requirements documents standards have been designed, e.g., IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

Requirements Document Structure

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system’s functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

Requirements Document Structure

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Requirements Validation

Requirements Validation

- Validity. Does the system provide the functions which best support the customer’s needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements Review

- Verifiability
    - Is the requirement realistically testable?
- Comprehensibility
    - Is the requirement properly understood?
- Traceability
    - Is the origin of the requirement clearly stated?
- Adaptability
    - Can the requirement be changed without a large impact on other requirements?

In-Class Activity: 25 minutes

1. Write at least 10 functional requirements for your project idea
2. Group them using Low - Medium - High
3. Discuss your project requirement with at least another group.
4. Ask other groups to suggest additional features or requirements.

Questions

References

- Chapter 4, Software-Engineering-10th-Ian-Sommerville
- Chapter 2, The IEEE Guide to the Software Engineering Body of Knowledge
- Part 2, Software Architecture in Practice, 3rd edition, by Bass, Clements, and Kazman