2. Analysis and specification of requirements

2.1. Requirements analysis and specification

Abstract

One of the first stages in software development is focused on the analysis and specification of the requirements that users have for the software, that is, the requirements that the future software must realize. Well-defined requirements are the basis of any software, because they indicate that we have a good understanding of the problem itself, but also that we have clear expectations from the future software solution. If the requirements are incorrectly or poorly defined, not even the best tools will help us create user-friendly software. The field, but also the process that deals with the collection, analysis, specification, and verification of requirements is called requirements engineering. In this lesson, we will cover the basic concepts associated with requirements engineering, with a particular focus on creating a structured software requirements specification.

Introduction

Requirements are the basis of every software. They describe what the software should be able to do in order to fulfill the needs of the software users. Well-defined requirements indicate that we understand the problem domain, that we are aware of the needs and expectations of the users, and that we have a clear vision of what the software should do in order to be useful. Research shows that clearly defined requirements are one of the most significant factors in the success of software projects. Already at the very beginning of a software project, we need them in order to be able to carry out feasibility assessment, project planning, risk management, assigning and prioritizing tasks, defining software acceptance criteria, etc. On the other hand, according to the same research, incomplete and unrealistic requirements are some of the of the most common reasons why software projects fail. Namely, if the software does not meet the needs of the users, it will not even have users, rendering it useless. Unfortunately, this cannot be corrected neither by using the most modern tools and technologies, nor by applying best practices in software design, implementation, or testing. Despite the critical importance of requirements, requirements management is often not given enough attention.

In traditional software development processes (eg. the waterfall model), the first phase of the software process is reserved for defining requirements. Nevertheless, requirements are by no means isolated from other traditional phases and activities of the software process. On the contrary, they guide the activities of making design and implementation decisions, as well as defining scenarios for software testing. There is often a feedback loop, so the implementation of design, implementation and testing activities result in new insights and identifying of new or changing requirements.

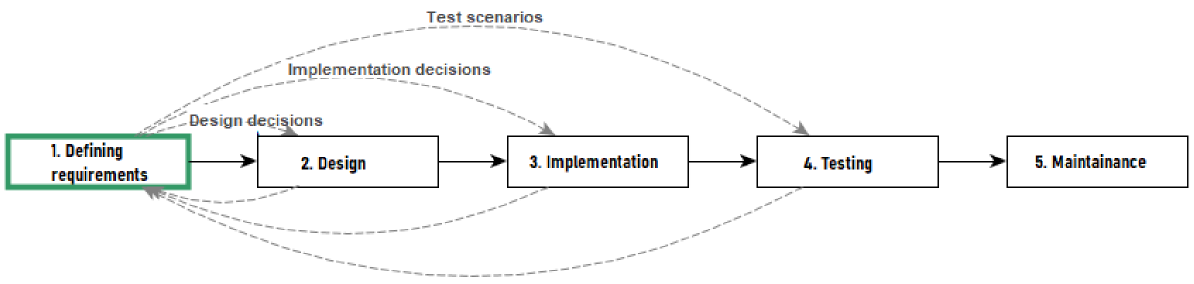

Figure 1: The role of requirements in the software development process

Defining requirements is an integral part of the software development process, even when it is not explicitly, or even consciously, performed and documented as an activity. In most cases, however, the requirements are explicitly defined with a lower or higher level of detail, and in a more or less formal and sophisticated manner. The area and process that deals with the systematic collection, analysis, specification, and verification of requirements is called requirements engineering. Although we view it as an area within software engineering, requirements engineering is not reserved exclusively for the context of software construction, but deals with system requirements in general.

Requirements process

Requirements elicitation

Elicitation is an activity that aims to identify the requirements that a future software solution should fulfill. Good communication with software stakeholders is extremely important in this activity, because they and their needs are the main source of requirements and limitations of the software solution. Stakeholders are individuals, groups or organizations that have a direct or indirect interest in the software solution. These can be all those who, due to the success or failure of the software solution, can realize profit or loss (not necessarily financial), and all those who are responsible for the creation and use of the software. This includes: managers who will approve and manage the software development project, investors who will finance the software development, users who will use the software, technical staff who will install and service the software, educators who will ensure the correct use of the software, etc. Due to their involvement in the software development process itself or the subsequent use of the software, stakeholders are one of the main sources of software requirements. We can collect data from stakeholders by conducting interviews, questionnaires, focus groups, workshops, and observing users in a business environment. Some of the problems that can arise are, for example, that stakeholders are not fully aware of their needs, that they have difficulty articulating their needs, and have unrealistic or conflicting demands.

In addition to stakeholders, we can reach requests by analyzing the market, business documentation, software solutions in use or similar software solutions, reported problems and suggestions related to existing software solutions, prototypes, etc.

Requirements specification

Once the requirements that the future software solution needs to fulfill have been identified, they need to be documented in a clear, easy-to-understand, consistent and unambiguous manner. This is exactly the goal of specifying requirements as an activity. When documenting requirements, i.e. creating a specification, we usually rely on the use of natural language in order to have flexibility and richness of expression. However, to ensure that the requirements specification is correctly structured and contains all the necessary information, we rely on various guidelines (e.g., EARS or MOSCOW) and ready-made templates (e.g., Volere, or IEEE 830-1998 SRS). In addition to natural language, approaches with a higher level of structure (e.g., UML diagramming techniques), and formal and mathematical approaches (e.g., VDM or Z language) can be used to specify requirements.

Checking the request

One of the activities we carry out when checking requests is the so-called request *validation.* It aims to determine whether the specified requirements reflect the needs and expectations of the stakeholders, and whether the requirements are specified in a correct way (e.g., whether they are clear, precise, feasible, etc.). Validation will ensure that we have the correct set of requirements that will result in custom software. If we fail to validate the requirements in the early stages of the software development process, the cost of correcting an incorrect requirement in the later stages will be many times higher. The final validation of the specified requirements will be carried out by the users themselves before handing over the developed software.

The verification activity is also important for checking requests. It aims to verify whether the later stages of the software development process have fully and adequately realized the specified requirements. In other words, verifications should show that all specified requirements were taken into account during software design, that they were adequately implemented and covered by tests.

For this purpose, the so-called traceability matrix is created. It maps each requirement from the specification to other software artifacts such as UML diagrams, program code, and test scenarios.

Table 1: Monitoring matrix

Requirement	Design	Implementation	Testing
FZ-1	UML: UC#1, DC#1, DSeq#1	Class X, method X1, component 2	Unit Tests: #1-#12 Acceptance tests: AT#1,#2
FZ-2	UML: UC#1, DAct#1, DC#2	Class Y, method Y1, component 2	Unit Tests: #14-#18 Acceptance Tests: AT#3-#5
…	…	…	…

Defining the structure of the requirements specification

Key results of defining requirements are the user requirements specification (USR) and software requirements specification (SRS) documents. The user requirements specification contains requirements of a higher level of abstraction that describe the user's needs and expectations that the user has towards the system. It is usually written in natural language using terminology from the problem domain. The use of technical terms is kept to a minimum in order to make the document understandable to the client, end users, managers and other non-technical stakeholders who together represent the target group of readers of the document. The specification of user requirements itself can be created by the software supplier based on analyzed documentation, market research and communication with stakeholders. However, often (and especially in highly regulated domains such as the public sector) the client himself creates the specification of user requirements if he has technical knowledge. A software requirements specification expands on what is specified in user requirements to describe in detail what the software should do to meet the needs and expectations of users. Often, multiple software requirements are defined based on one user request. Although the specification of software requirements is usually written in natural language, given that it is mainly aimed at technically savvy stakeholders (programmers, software architects, testers, etc.), the use of technical terminology and diagrammatic techniques is not avoided. At the same time, it is necessary to take care that the specification of software requirements is focused on defining what the software should do, not how. In other words, information related to software design and implementation should not be part of the software requirements specification.

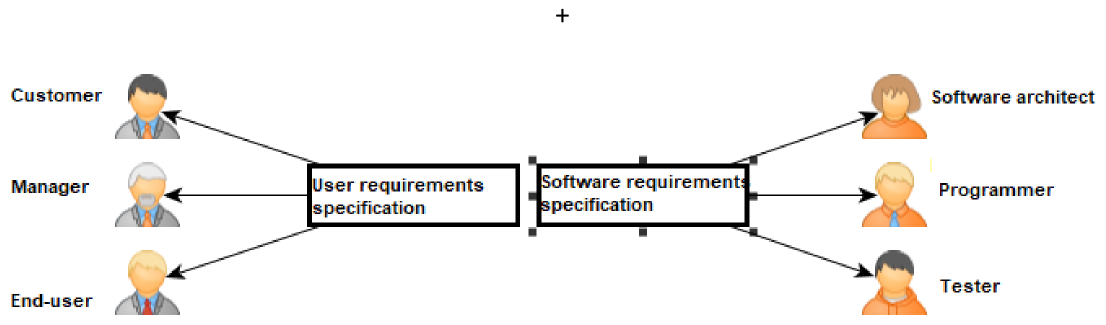
Customer
Manager
End-user
User requirements specification
Software requirements specification
Software architect
Programmer
Tester

Figure 2: Target groups of user and software requirements specifications

In order for the requirements specification to be made with quality, it should contain the information needed by the target audience, and organize this information into a consistent and well-structured document. Such organization of requests and their related information helps in better understanding of requests, minimizing the number of requests, identifying duplicates, eliminating conflicts between requests and verifying the completeness of a set of requests. For the organization and structuring of the requirements specification document, templates (in the original or adapted for your own use) such as the Volere template or the IEEE 830-1998 SRS template are very often used.

Template based on IEEE 830-1998 document

For the purposes of this lesson, we have defined a software requirements specification template based on the IEEE 830-1998 document. All template chapters and instructions on how to write them can be found in the following table.

Table 2: Template for creating a requirement specification

No.	Title of the chapter	Instructions
1.	**Introduction**	
1.1.	Purpose	Describe the purpose of the SRS document (not the software solution). Specify who the document is intended for, i.e. who should read and understand the document.
1.2.	Extent	Briefly explain the problem domain that the solution addresses. In what context will the software solution be used? Give a name to the software solution, which will be used to refer to it in the rest of the document, and, if necessary, the version. Explain whether this is a completely new solution or an upgrade. Specify what the software will and will not do. What benefits and improvements do we expect the software solution to bring.
1.3.	Definitions, acronyms and abbreviations	Define the basic terms, acronyms, and abbreviations used in the specification that are necessary to properly understand the SRS document.
1.4.	References	Reference all documents, websites, standards, other specifications mentioned in the SRS.
1.5.	Document structure	Briefly describe how the rest of the document is organized and what it contains.

2.	**General description**	
2.1.	Product perspective	Put the software solution in context and relationship with other related systems. Indicate whether the software solution is independent, a part of a larger system (which is often the case) or is a replacement for an existing system. If it is part of a larger system, it is necessary to compare the requirements of the entire system with the requirements of our software solution, and clearly define the interfaces between them. In addition to the relationship with any superior system, also list the external systems and software that your software solution uses. These can be, for example: DMBS, operating system, web services and APIs . In case the software solution directly uses hardware and communication technologies, describe the features of the interface with them (type of hardware, port, communication protocols, bluetooth, NFC, IC, TCP, data exchange format, etc.).
2.2.	Product Features	List the main functions that the software solution should contain, without going into the details of each of these functions (detailed requirements will be contained in Chapter 3). The functions should be organized and described in such a way that they are understandable to all readers of the document, including the client/user, on the first reading. For the formulation of the functions of the software solution at this level, a higher-level specification can be used, for example, a specification of user requirements (if any).
2.3.	User characteristics	Identify user groups/roles that we expect to use the software solution (eg teacher, professor, administrator, cashier, bank clerk, accountant, …). State the general characteristics of the identified users, state how they differ. This may include the level of education, experience, computer and technical literacy, then the expected frequency of use of the software solution, the level of permissions granted, the set of functions of the software they will use.
2.4.	Limitations	List the aspects of the problem domain and the solution itself that can act as a limitation on the development of the software solution. This may include: legal and corporate rules and regulations (e.g., GDPR provisions, fiscalization, security policies, etc.); hardware limitations (e.g., in development for small devices – battery, memory, processor, connectivity, etc.); the need to adapt to other systems (eg interaction with existing and potentially outdated systems); security criticality of the application (e.g., in case the application controls dangerous machines, medical devices, or works with sensitive and confidential data, etc.); requirements for reliability (e.g., it may be required to use a certain approach, practice tools and standards in programming, modeling and testing of software solutions, etc.)
2.5.	Assumptions and dependency	List assumptions and open questions (as opposed to known facts) whose subsequent outcome may affect the requirements stated in this document. These are very often circumstances that are not under our responsibility (e.g., if there is a probable possibility or announcement of a change in legislation that will result in the modification of defined requirements; or the release of a new version of the API on which our software depends).
2.6.	The rest	List other aspects of the problem domain and the future software solution that you consider important and which are not provided for in other parts of the document.
3.	**Functional requirements**	Define functional requirements for software solutions in such a way as to provide enough information to designers and developers to begin designing and implementing solutions, and to testers to design test cases. The requirements listed here are based on the product functions described in section 2.2, but are described with a higher level of detail.

		The focus is on the function and limitations of the system. Each request should be assigned a unique identifier. Requests can also be grouped according to different criteria, such as users (teacher, student, cashier, administrator...), case of use, domain concepts, etc. The following table must be filled in for each request:

Identifier	Unique request identifier.
Claim[1]	Description of requirement in the form: " *The system will enable <functionality> <object> with <limitations>* ". The writing style should be uniform. When formulating, follow accepted guidelines for defining requests, such as those listed in the *INCOSE Guidelines for Writing Requests* .
Explanation	Reasoning why the request exists/why it is needed.
Verification method	A verification criterion or a test scenario that will allow determining whether a requirement is met or not.
Priority [1-5]	Request priority (1 – highest priority, 5 lowest priority)
Source/Origin	The name of the document by which the request is prescribed or of the stakeholder who submitted the request.

3.1.	Dynamics of request realization	Specify whether all requirements will be implemented in the initial version of the software, or whether some will be left for future versions. State if there are any other requests that are planned to be implemented in the future.
4.	**Non-functional requirements**	
4.1.	Software layout	List the requirements (if any) related to the appearance and visual style of the software. This may include requirements to use, for example, a formal and corporate style (in the case of a business application), or a playful style (e.g., a computer game for children); a request to use a specific color palette or controls so that the application is consistent with the company's branding, etc.
4.2.	Usability of software	List the requirements (if any) related to ease of learning and use of the software, adaptation for people with disabilities, localization. This may include requirements related to the learning curve, the speed of using the application (eg speed of data entry), the ease of remembering software options, the frequency of errors that the user makes when working with the software, the ability to select a language, the ability to be used by blind people, etc.
4.3.	Software performance	List the requirements (if any) related to the performance of the software. This may include requirements related to task processing speed, responsiveness, precision of results, storage capacity, scalability, system availability, etc.
4.4.	Software execution and environment	List the requirements (if any) associated with running the software and the environment in which the software runs. This may include requirements related to the physical environment in which the system is located (loud environment, strong lighting, dust, etc.), other existing

[1] Alternatively, this could be called "Requirement" or "Name".

		systems within which the software should run or interact with (operating system, other software from which we download data or we send them).
4.5.	Security and privacy	List the requirements (if any) related to issues of data security and privacy, as well as standards and regulations related to these issues. This may include requirements for the use of prescribed security procedures, technologies, compliance with legal frameworks, etc.
4.6.	The rest	List other non-functional requirements not listed above.
5.	**Screenshots**	Visualize the features of the interaction between the end user and the software solution through wireframes. The purpose of sketches is to visually represent and communicate what the application should do, not to create a realistic design of the graphical interface. At the same time, you can draw sketches in any way (e.g., manually on paper + drawing with a mobile phone, MS Paint, MS Word, Excel...)

A practical example

This subsection demonstrates the creation of a software requirements specification document on a practical example of software for recording the results of the implementation of continuous monitoring in the Software Engineering course. The practical example begins with a brief description of the problem domain given by the client (teacher on the course), which enables potential suppliers of the software solution (students on the course) to become familiar with the problem domain and user needs. Based on the described problem domain, the software requirements specification document is created. The very structure of the document is taken from the template defined in the previous chapter. It should be noted that in this lesson we are creating only the first two chapters of the specification, i.e. Introduction and General Description. The remaining part of the software requirements specification, which includes functional and non-functional requirements, will be covered in the next lesson.

Description of the problem domain

During the teaching of the Software Engineering course, the teachers, as part of the continuous monitoring of the Software Engineering course (2nd year, IPS field of study), evaluate the students according to several monitoring elements. Considering the number of students (expected around 200) and the number of monitoring elements (currently 5), the number of individual evaluations reaches 1000. Each of these evaluations results in the number of points that teachers must record in order to be able to give a final evaluation to the student at the end of the semester.

Teachers currently keep records using a spreadsheet calculator, however, due to a number of shortcomings and problems that arise, they would like to switch to specially designed software. Some of the problems are: the demanding procedure of setting up a table calculator for each academic year, certain actions are difficult to automate, lack of automated reports, complex adjustment in case of changes in the monitoring model, etc.

Since the data related to the evaluation results are sensitive (private), it is definitely necessary to limit access to the records so that only teachers can access them. The very elements of monitoring and the conditions that students must meet, teachers would like to be able to define in the event of a need to change the monitoring model. In the same case, it should be possible to define a point scale.

At the beginning of the academic year, we definitely want to register the students who are enrolled in the course. At the same time, it is desirable to have the possibility of collective and faster entry of students.

Teachers want to be able to record students' points, for each defined monitoring element, where it must be known which teacher entered the points. And at the end, based on the achieved points and the point scale, we want to get a proposal for the grade for the student. Of course, this grade is only a suggestion, and the teacher himself enters the grade he considers correct.

Given that before the deadline for continuous monitoring opens, it is necessary to record the prohibition of signatures in the ISVU system, teachers need a list of students who have not met the conditions for signature. And finally, for the purposes of recording the grade/implementation of the oral exam during the period for continuous monitoring, teachers need a display of the detailed results of continuous monitoring for all students who have exercised the right to sign.

The current monitoring model of the Software Engineering course prescribes a total of 5 monitoring elements (see Table 3). For each monitoring element, the maximum number of points that can be achieved, the number of points that must be achieved as a condition for the signature[2], and the number of points that must be achieved for the evaluation are defined. As can be seen, the 1st and 2nd theoretical colloquiums (written theoretical exams) are not conditions for the signature, while all three tasks are. On the other hand, all five monitoring elements are a condition for evaluation.

Table 3: Monitoring elements of the Software Engineering course

Monitoring elements	Max. points	Signature condition	Condition for evaluation
1st theoretical colloquium	25	0	11
2nd theoretical colloquium	25	0	11
Task 1	15	6	6
Task 2	15	6	6
Task 3	20	8	8
In total	**100**		

In case the student achieves at least the minimum required number of points from each monitoring element, the grade is proposed based on the total number of points achieved using the following point scale:

Table 4: Point scale for the Software Engineering course

Points from	Points up to	Evaluation
0	49	1
50	60	2
61	75	3
76	90	4
91	100	5

[2] In Croatian academic system, getting a „signature“ represents passing the course. It used to be an actual signature from the professor for decades, but now it’s just a digital “tick” in the system. Without all the conditions for a signature, you undertake the entire course next year – even if you achieved 50+ points.

Software requirements specification for Evaluation Manager

1. INTRODUCTION

1.1. Purpose

This document represents the specification of software requirements for a software intended for recording the results of the implementation of continuous monitoring in the Software Engineering course. The requirements specification was created based on the initial user requirements submitted by the course teacher. The target group of the requirements specification is project managers who will manage the dynamics of solution creation, designers and programmers who need to design and implement a software solution, and testers who will verify that the solution truly meets the set requirements. In addition to being a basis for further development of the software solution, this document also has the purpose of a contract between the client (teacher of the Software Engineering course) and the contractor (selected company), so the target group of readers includes clients.

The structure of the document itself is based on the template defined in the document *IEEE 830-1998 Recommended Practice for Software Requirements Specifications*.[3]

1.2. Extent

When teaching the Software Engineering course, teachers evaluate students according to defined monitoring elements within the framework of continuous monitoring of the Software Engineering course (2nd year of FOI, IPS field of study). Given the number of enrolled students and the number of defined monitoring elements, there are often more than 1,000 individual evaluations. During each evaluation, the teacher records the number of points the student has achieved. At the end of the continuous monitoring, the points obtained from the individual elements of the monitoring are added up, and it is checked whether the student has achieved the right to sign and grade.

Table 5: Monitoring elements of the Software Engineering course

Monitoring elements	Max. points	Signature condition	Condition for evaluation
1st theoretical colloquium	25	0	11
2nd theoretical colloquium	25	0	11
Task 1	15	6	6
Task 2	15	6	6
Task 3	20	8	8
In total	**100**		

[3]http://ieeexplore.ieee.org/servlet/opac?punumber=5841

The current monitoring model of the Software Engineering course prescribes a total of 5 monitoring elements (see Table 5). For each monitoring element, the maximum number of points that can be achieved, the number of points that must be achieved as a condition for a signature, and the number of points that must be achieved for the evaluation are defined. It is considered that the student has not achieved the condition for the signature or evaluation if he has not achieved the number of points sufficient for signature or evaluation from any monitoring element. As can be seen, the 1st and 2nd theoretical colloquium are not conditions for the signature, while all three assignments are. On the other hand, all five monitoring elements are a condition for a positive evaluation/grade.

In the event that the student exercises the right for the signature, the grade is proposed based on the total number of achieved points using the following point scale:

Table 6: Point scale for the Software Engineering course

Points from	Points up to	Evaluation
0	49	1
50	60	2
61	75	3
76	90	4
91	100	5

Teachers currently keep records using a spreadsheet calculator, however, due to a number of shortcomings and problems that arise, they would like to switch to specially designed software. Some of the problems are: the demanding procedure of setting up a table calculator for each academic year, certain actions are difficult to automate, lack of automated reports, complex adjustment in case of changes in the monitoring model, etc.

Considering the described problem and user expectations, it is proposed to create a new software solution called Evaluation Manager, which would replace the existing record of continuous monitoring results created in a spreadsheet calculator. It should be emphasized that Evaluation Manager includes only continuous monitoring, i.e. records the results of only those students who have chosen the full-time student monitoring model (so-called Model A). Monitoring of students who are not part of continuous monitoring, i.e. those who have chosen the monitoring model of part-time students (so-called Model B) is not foreseen by this software solution. Also, the software solution does not include regular or extraordinary exam periods after the period for continuous monitoring.

1.3. Definitions, acronyms and abbreviations

- *Monitoring model* - the model according to which the student attends classes during the semester. It defines the responsibilities and activities that the student should perform, and the way in which the performed activities will be evaluated.
- *Monitoring element* – an individual activity that the student should do independently or in a team, and which is scored by the teacher (e.g. colloquium/exam, assignment).
- *Condition for signature* – criteria for a particular monitoring element that must be met in order for the student to exercise the right to get a signature from the course.
- *Condition for assessment* – a criterion for an individual monitoring element that must be met in order for the student to exercise the right to a grade from the course.
- *ISVU* – Information system of higher education institutions that the faculty uses as support during the implementation of business processes related to teaching.

1.4. References

1. "830-1998 - IEEE Recommended Practice for Software Requirements Specifications ." IEEE, 1998 [Online]. Available : http://ieeexplore.ieee.org/servlet/opac?punumber=5841
2. Problem domain description document
3. Course monitoring model Software engineering, 2022, [Online]. Available : https://nastava.foi.hr/course/214467

1.5. Document structure

In *chapter 2* We put Evaluation Manager in context and describe the interaction with users but also with other systems, software solutions, hardware, and communication technologies. Next, we briefly describe the basic functions that the Evaluation Manager will perform, the characteristics of the users who will use the software, and the limitations that may affect the development of the software solution itself.

In *Chapter 3,* we define the functional requirements for Evaluation Manager at a level of detail that is sufficient for designers and developers to begin designing and implementing solutions, and for testers to design test cases.

In *Chapter 4,* we define the non-functional requirements for Evaluation Manager that designers and developers should consider when designing the architecture, choosing implementation technologies and approaches.

In *Chapter 5,* we visualize how users interact with Evaluation Manager by sketching a graphical user interface.

2. GENERAL DESCRIPTION

2.1. Product perspective

Evaluation Manager is designed as an independent software solution that is a replacement for the existing system (based on a spreadsheet) of recording the results of continuous monitoring. The software solution should contain a client application that will run on the end user's computer, while the database would be centralized due to the need to share data between teachers. Direct interaction with other systems that are not an integral part of Evaluation Manager is not foreseen. Since the import of data on enrolled students from the ISVU system is foreseen, there is an indirect dependence on the ISVU system with regard to the file format and data that ISVU can deliver.

Evaluation Manager does not require the direct use of hardware or communication technologies. Any use of such resources is intended to be handled by the operating system or runtime environment.

2.2. Product Features

Future users of the Evaluation Manager software solution expect the following features from the software:

- Limiting access to records.
- Defining the course monitoring model.
- Entry of students enrolled in the course.
- Recording of the student's achieved points.
- Suggesting a grade for a student.
- Printing the results of continuous monitoring.

2.3. User characteristics

The users who will use the Evaluation Manager software solution are teachers of the Software Engineering course. Given that all teachers will use the software in the same way and have the same level of rights, we can say that there is one user role - *teacher.* However, each teacher will be asked to log into the application before use with their own user information, so that we can distinguish what results each teacher has recorded. All teachers have an advanced level of computer and technical literacy.

2.4. Limitations

Given that the Evaluation Manager software solution involves working with students' private results (achieved results on continuous monitoring) that are subject to GDPR provisions, it is necessary to ensure that only authorized persons (course teachers) have access to said data. With regard to the problem domain and the characteristics of the solution, the existence of additional restrictions was not observed: it is not a security-critical domain; interaction with other systems is not direct and does not significantly affect the development of the software solution itself; the hardware characteristics of today's computers are more than sufficient for working with the software. The contractor is expected to develop the Evaluation Manager software solution in accordance with the good practices of the profession, however, the client does not set specific restrictions regarding the methodological approach, tools, and production technology.

2.5. Assumptions and dependencies

The model for monitoring students' work within the framework of continuous monitoring is predefined and is not subject to changes in the current academic year, therefore no extraordinary changes to the requirements are expected. Also, during the development period of the Evaluation Manager software solution, no such changes are expected in the selected technology that would cause changes at the level of software requirements.

2.6. The rest

There is no need to elaborate additional aspects.

Questions

1. What is a requirement?

2. What does requirements engineering do?

3. Who/what is a stakeholder in the context of software requirements?

4. What are the basic activities of the requirements definition process?

5. What is the difference between user requirements and software requirements?

6. Why is it important to specify requirements?

7. How are requirements related to other activities of the software development process?

8. Why is it important to have a structured requirements specification document?