ICS 3105
OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter 5.0

Developing Requirements

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Define customer's problem and setting the scope for the project.
  - Identify various types of requirements.
  - Requirements documents and what should be put in them.

Learning Outcomes

- Gather requirements.
- Model users' tasks using use case diagrams and detailed descriptions of use cases.
- Review a set of requirements.

Domain Analysis

- Domain analysis is the process by which a software engineer learns background information.
- He or she has to learn sufficient information so as to be able to understand the problem and make good decisions during requirements analysis and other stages of the software engineering process.

Domain Analysis

- The word ‘domain’ in this case means the general field of business or technology in which the customers expect to be using the software.
- Some domains might be very broad, such as ‘airline reservations’, ‘medical diagnosis’, and ‘financial analysis’.

Domain Analysis

- Others are narrower, such as 'the manufacturing of paint' or 'scheduling meetings'.
- People who work in a domain and who have a deep knowledge of it (or part of it) are called domain experts.
- Many of these people may become customers or users.

Domain Analysis

- To perform domain analysis, you gather information from whatever sources of information are available.
- These sources include the
  - Domain experts;
  - Any books about the domain;
  - Any existing software and its documentation; and any other documents you can find.

Domain Analysis

- The interviewing, brainstorming and use case analysis techniques help with domain analysis.
- As a software engineer, you are not expected to become an expert in the domain; nevertheless, domain analysis can involve considerable work.

Benefits of Domain Analysis

- Faster development.
- Better system.
- Anticipation of extensions.

Faster development

- You will be able to communicate with the stakeholders more effectively, hence you will be able to establish requirements more rapidly.
- Having performed domain analysis will help you to focus on the most important issues.

Better system

- Knowing the idea of the domain will help ensure that the solutions you adopt will more effectively solve the customer's problem.
- You will make fewer mistakes, and will know which procedures and standards to follow.

Better system

- The analysis will give you a global picture of the domain of application; this will lead to better abstractions and hence improved designs.

Anticipation of extensions

- Armed with domain knowledge, you will obtain insights into emerging trends and you will notice opportunities for future development.
- This will allow you to build a more adaptable system.
- It is useful to write a summary of the information found during domain analysis.

Anticipation of extensions

- The process of organizing and writing this summary can help you gain a better grasp of the knowledge; the resulting document can help educate other software engineers who join the team later.

Sections of domain analysis document

- Introduction.
- Glossary.
- General knowledge about the domain.
- Customers and users.
- The environment.
- Tasks and procedures currently performed.
- Competing software.
- Similarities across domains and organizations.

Introduction

- Name the domain, and give the motivation for performing the analysis.
- The motivation normally is that you are preparing to solve a particular problem by development or extension of a software system.

Glossary

- Describe the meanings of all terms used in the domain that are either not part of everyday language or else have special meanings.
- You must master this terminology if you want to be able to communicate with your customers and users.

Glossary

- The terminology is likely to appear in the user interface of the software as well as in the documentation.
- You may be able to refer to an existing glossary in some other document, rather than writing a new glossary.

Glosary

- The section is best placed at the start of the domain analysis document so that you can subsequently use the defined terms.

General knowledge about the domain

- Summarize important facts or rules that are widely known by the domain experts and which would normally be learned as part of their education.
- Such knowledge includes scientific principles, business processes, analysis techniques, and how any technology works.

General knowledge about the domain

- This is an excellent place to use diagrams; however, where possible, point the reader for details to any readily accessible books or other documents.
- This general knowledge will help you acquire an understanding of the data you may have to process and computations you may have to perform.

Customers and users

- Describe who will or might buy the software, and in what industrial sectors they operate.
- Also, describe the other people who work in the domain, even peripherally.
- Mention their background and attitude as well as how they fit into the organization chart, and relate to each other.

The environment

- Describe the equipment and systems used.
- The new system or extensions will have to work in the context of this environment.

Tasks and procedures currently performed

- Make a list of what the various people do as they go about their work.
- It is important to understand both the procedures people are supposed to follow as well as the shortcuts they tend to take.

Tasks and procedures currently performed

- If, for example, people are supposed to enter certain information on a form, but rarely do so, this suggests that the information is not useful.
- Tasks listed in this section may be candidates for automation.

Competing software

- Describe what software is available to assist the users and customers, including software that is already in use, and software on the market.
- Discuss its advantages and disadvantages.
- This information suggests ideas for requirements, and highlights mistakes to avoid.

Similarities across domains and organizations

- Understanding what is generic versus what is specific will help you to create software that might be more reusable or more widely marketable.
- Therefore, determine what distinguishes this domain and the customer's organization from others, as well as what they have in common.

Domain Analysis

- Be careful not to write an excessive amount of detailed information.
- It is a waste of effort to duplicate the original source material; your domain analysis should simply include a brief summary of the information you have found, along with references that will enable others to find that information.

Domain Analysis

- No serious software project should be undertaken without a sound domain analysis; a good knowledge of the domain of application considerably increases your chances of success.

Domain Analysis

- Many of the most successful software products have been developed by people who were actively working in the domain before they became software developers – such people have a better feel for what is really needed.

Domain Analysis

- Once software engineers have a good grasp of the domain, they can move on to requirements analysis, which includes defining the problem to be solved and what software will be created to solve it.

Domain Analysis

- However, domain analysis should never really end: software engineers have the responsibility to continue improving their understanding as development proceeds.
- An extension to the system added for a subsequent release will often merit a domain analysis of its own subdomain.

The starting point for software projects

- When a development team starts work on a software project, their starting point can vary considerably.
- We can distinguish different types of project, based on whether or not software exists at the outset, and whether or not requirements exist at the outset.

Starting points for software projects

	Requirements must be determined	Clients have produced requirements
New development	A	B
Evolution of existing system	C	D

Starting points for software projects

- In projects of type A or B, the development team starts to develop new software from scratch – this is sometimes called green-field development, alluding to constructing a new building where none existed before.
- In cases C and D the team evolves an existing system, a rather more common situation.

Starting points for software projects

- In cases A and C, the development team has to determine the requirements for the software – they either have a bright idea for something that might sell, or else they are asked to solve a problem and have to work out the best way to solve it.

Starting points for software projects

- In cases B and D, on the other hand, the development team is contracted to design and implement a very specific set of requirements.
- In these latter cases, the customer's organization has normally done the requirements analysis, perhaps using in-house software engineers or consultants specializing in requirements analysis.

Starting points for software projects

- Projects where the requirements are pre-specified should be handled carefully.
- If the customer has not done a good job of analysis and specification, the requirements are likely to be poor.
- For example, the customer may be proposing a system that is far too large, or that does not address a clear problem.

Starting points for software projects

- As a software engineer, you have a professional responsibility to ensure that the requirements on which you base your work are of good quality, even when they were developed by others.

Starting points for software projects

- You should evaluate such requirements yourself, and work with the customers to resolve any problems.
- You should not accept a contract where you are required to implement requirements with no changes allowed.

Defining the problem and the scope

- Once you have learned enough about the domain, you can begin to determine the requirements.
- The first step in this process is to work out an initial definition of the problem to be solved.

Defining the problem and the scope

- A problem can be expressed as a difficulty the users or customers are facing, or as an opportunity that will result in some benefit such as improved productivity or sales.

Defining the problem and the scope

- The solution to the problem will normally entail developing software, although you may decide that it is better to purchase software or to develop a non-software solution.

Defining the problem and the scope

- You should write the problem as a simple statement.
- Careful attention to the problem statement is important since, later on, the requirements will be evaluated based on the question: 'are we adequately solving the problem?'

Defining the problem and the scope

- A good problem statement is short and succinct – one or two sentences is best.
- For example, if you were developing a new student registration system, you might express the problem as follows: 'The system will allow students to register for courses, and change their registration, as simply and rapidly as possible. It will help students achieve their personal goals of obtaining their degree in the shortest reasonable time while taking courses that they find most interesting and fulfilling.'

Defining the problem and the scope

- If the problem is broad, or contains a long list of sub-problems, then the system will have a broad scope, and hence be more complex.
- An important objective is to narrow the scope by defining a more precise problem.

Defining the problem and the scope

- In the above example, if we had stated: 'the system will automate all the functions of the registrar's office', that leaves open the possibility of including such features as fee payment, printing class lists and allocating rooms to courses.

Defining the problem and the scope

- One way to set the scope is to list all the sub-problems you might imagine the system attacking.
- To narrow the scope, you can then exclude some of these sub-problems – perhaps they can be left for another project.

Defining the problem and the scope

- Sometimes, an inappropriate choice of problem statement can result in a scope that is too narrow or completely wrong.
- To determine whether this is the case, think about what will be the user's ultimate high-level goal when they use the system, and the customer's high-level goal for having it developed.

Defining the problem and the scope

- In the university registration example you could consider a student's goal to be 'completing the registration process'.
- However, you can see that the student's higher-level goal might be, 'obtaining their degree in the shortest reasonable time while taking courses that they find most interesting and fulfilling'.

Defining the problem and the scope

- This new goal sheds a different light on the problem; you might consider adding features to the system that would not otherwise have occurred to you, such as actively proposing courses based on an analysis of the student's academic and personal-interest profiles.

Defining the problem and the scope

- All the requirements gathering and analysis techniques can help in defining the problem and hence the system's scope.
- Interviewing can give you the stakeholders' personal perspectives; brainstorming can generate lists of ideas from which you can extract a suitable problem or problems; use case analysis can give you a list of the possible things the system could do; and prototyping can give everybody a better perspective about what might be possible.

Defining the problem and the scope

- It is a good idea to define the problem and scope as early as possible, before getting deeper into analysis of the detailed requirements.
- This prevents you from working on unnecessary requirements.

Defining the problem and the scope

- However, as with domain analysis, your perspective on the problem will improve as analysis continues, hence the problem statement may need to be refined several times.

What is a requirement?

- A requirement is a statement describing either:
  - 1) an aspect of what the proposed system must do, or
  - 2) a constraint on the system's development.

Requirements

- In either case, it must contribute in some way towards adequately solving the customer's problem; the set of requirements as a whole represents a negotiated agreement among all stakeholders.

Requirements

- A *requirement is a statement* i.e. this means that each requirement is a relatively short and concise piece of information, expressed as a fact.
- It can be written as a sentence or can be expressed using some kind of diagram.
- A collection of requirements is called a requirements document.

Requirements

- *an aspect of what the proposed system must do…*: most requirements say something about the tasks the system is supposed to accomplish.
- They do not describe the domain.

Requirements

- *a constraint on the system's development…*: requirements often specify the quality levels required.
- They may also specify details such as the programming language to be used if this is truly important to the customer.
- They should, however, avoid discussing incidental aspects of the design.

Requirements

- *…contribute … towards adequately solving the customer's problem*: a requirement should only be present if it helps solve the customer's problem.

Requirements

- *…a negotiated agreement among all stakeholders…*: a statement about the system should not be declared to be an official requirement until all the stakeholders (users, customers, developers and their management) have reviewed it, have negotiated any needed changes, and have agreed that it is valid.

Types of requirements

- Requirements can be divided into four major types:
  - Functional,
  - Quality,
  - Platform and
  - Process.

Types of requirements

- Requirements documents normally include at least the first two types.

Types of requirements: Functional requirements

- Functional requirements describe what the system should do; in other words, they describe the services provided for the users and for other systems.
- The

Types of requirements

- The functional requirements should include:
  1) Everything that a user of the system would need to know regarding what the system does, and
  2) Everything that would concern any other system that has to interface to this system.
- Details that go any deeper into how the requirements are implemented should be left out.

Types of requirements: Functional Requirements

- The functional requirements can be further categorized as follows:
  - What inputs the system should accept, and under what conditions.
  - What outputs the system should produce, and under what conditions.
  - What data the system should store that other systems might use.
  - What computations the system should perform.
  - The timing and synchronization of the above.

Types of requirements: Functional Requirements

- What inputs the system should accept, and under what conditions.
  - This includes data and commands both from the users and from other systems.

Types of requirements: Functional Requirements

- What outputs the system should produce, and under what conditions.
  - Outputs can be to the screen or printed. They can also be transmitted to other systems, such as special I/O devices, clients or servers.

Types of requirements: Functional Requirements

- What data the system should store that other systems might use.
  - This is really a special kind of output that will eventually become an input to other systems. Data which is stored for the exclusive use of this system (e.g. the specifics of a file format used to temporarily back up some data) can be ignored until the design stage.

Types of requirements: Functional Requirements

- What computations the system should perform.
  - The computations should be described at a level that all the readers can understand. For example, you would describe a sorting process by saying that the result is to be ordered in ascending sequence according to the account number. You would not normally specify the particular algorithm to be used.

Types of requirements: Functional Requirements

- The timing and synchronization of the above.
  - Not all systems involve timing and synchronization – this category of functional requirements is of most importance in hard real-time systems that do such things as control hardware devices (e.g. telecommunications systems, systems that control power plants or factories, and systems that run automobiles and airplanes)

Functional Requirements

- An individual requirement often covers more than one of the above categories.
- For example, the requirements for a word processor might say, ‘when the user selects “word count”, the system displays a dialog box listing the number of characters, words, sentences, lines, paragraphs, pages, and words per sentence in the current document.’ This requirement clearly describes input (selecting ‘word count’), output (what is displayed) and computation (counting all the necessary information, and computing the average words per sentence).

Quality requirements

- Quality requirements ensure the system possesses quality attributes such as usability, efficiency, reliability, maintainability and reusability.
- These requirements constrain the design to meet specified levels of quality.

Quality requirements

- One of the most important things about quality requirements is to make them verifiable i.e. it should be possible, after the system is implemented, to determine whether they have, in fact, been adhered to.

Quality requirements

- The verification is normally done by measuring various aspects of the system and seeing if the measurements conform to the requirements.

Quality requirements

- The following are some of the main categories of quality requirements:
  - Response Time, Throughput, resource usage, reliability, availability, recovery from failure, allowances for maintainability and enhancement, allowances for reusability.

Quality Requirements: Response Time

- For systems that process a lot of data or use a network extensively, you should require that the system gives results or feedback to the user in a certain minimum time.
- For example, you might write that a result must be calculated in less than three seconds, or that feedback about the progress of a search must appear within one second.

Quality Requirements: Response Time

- However, that for hard real-time systems, response time requirements should be considered to be functional – the system would not work unless they are adhered to

Quality Requirements: Throughput

- For number-crunching programs that may take hours, or for servers that continually respond to client requests, it is a good idea to specify throughput, in terms of computations or transactions per minute.

Quality Requirements: Resource Usage

- For systems that use non-trivial amounts of such resources as memory and network bandwidth, you should specify the maximum amount of these resources that the system will consume.
- This allows others to plan hardware upgrades.

Quality Requirements: Resource Usage

- For example, you could specify that no more than 50 MB of memory is to be used by the system, and that the system must consume less than 10% of the CPU's time when run on a 1.8GHz machine under a certain operating system.

Quality Requirements: Reliability

- Reliability is measured as the average amount of time between failures or the probability of a failure in a given period.
- It is a good idea to set strong but realistic targets for this.

Quality Requirements: Reliability

- For example, you might specify that a continuously running server must not suffer more than one failure in a six-month period.

Quality Requirements: Reliability

- It is necessary to define what you mean by a failure: normally it means much more than just crashes; failures normally include any difficulties users have getting their work done which are attributable to defects.

Quality Requirements: Availability

- Availability measures the amount of time that a server is running and available to respond to users. As with reliability, you should set a target for this.
- For example, you might specify that a server must be available over 99% of the time, and that no period of downtime may exceed 1 minute.

Quality Requirements: Availability

- Telecommunications systems have very rigorous availability criteria: for example, you might specify that such a system must not be down more than 10 minutes in its 20-year life-span.
- This is also often called ‘6-nines’ availability, since it is equivalent to saying that the system must be up 99.9999% of the time.

Quality Requirements: Recovery from failure

- Requirements in this category constrain the allowed impact of a failure.
- They state that if the hardware or software crashes, or the power fails, then the system will be able to recover within a certain amount of time, and with a certain minimal loss of data.

Quality Requirements: Recovery from failure

- For example, the requirements for a word processor might state: 'the system will allow users to continue their work after a failure with the loss of no more than 20 words of typing or 20 formatting commands.'
- Note that the detailed procedure for recovery from failure is a functional requirement.

Quality Requirements: Allowances for maintainability and enhancement

- In order to ensure that the system can be adapted in the future, you should describe changes that are anticipated for subsequent releases.
- This constrains design and improves quality without adding explicit new functional requirements.

Quality Requirements: Allowances for reusability

- Similarly to the previous category, it is desirable in many cases to specify that a certain percentage of the system, e.g. 40%, measured in terms of lines of code, must be designed generically so that it can be reused.
- This will help break the reuse vicious cycle.

Platform requirements

- This type of requirement constrains the environment and technology of the system:
  - Computing platform.
  - Technology to be used.

Platform requirements: Computing Platform

- Computing platform.
  - It is normally important to make it clear what hardware and operating system the software must be able to work on. Such requirements specify the least powerful platforms and declare that it must work on anything more recent or more powerful.

Platform requirements: Computing Platform

- For example, you might declare that certain software must run on any computer operating under Mac OS X version 10.2 or MS-Windows 98 or higher, with 128 MB of RAM or more, and 100 MB of free disk space. This is quite different from the resource usage constraint on memory above: a resource usage constraint specifies that the system will not use more than a certain amount of resources, whereas a platform constraint says it is not guaranteed to run if inadequate or incorrect resources are available.

Platform requirements: Technology to be used

- Technology to be used.
  - While it is wise to give the designers as much flexibility as possible in choosing how to implement the system, sometimes constraints must be imposed. Common examples are to specify the programming language or database system.

Platform requirements: Technology to be used

- Such requirements are normally stated to ensure that all systems in an organization use the same technology – this reduces the need to train people in different technologies. The company might have also spent considerable money on a certain technology and wants to get the best value for that money.

Process requirements

- The final type of requirements constrains the project plan and development methods:
  - Development process (methodology) to be used.
  - Cost and delivery date.

Process requirements: Development Process

- Development process (methodology) to be used. In order to ensure quality, some requirements documents specify that certain processes be followed; for example, particular approaches to testing.
- The details of the process should not be included in the requirements; instead a reference should be made to other documents that describe the process.

Process requirements: Cost and delivery date

- These are important constraints.
- However, they are usually not placed in the requirements document, but are found in the contract for the system or are left to a separate project plan document.


JOMO KENYATTA UNIVERSITY
OF AGRICULTURE AND TECHNOLOGY
TECHNOLOGY FOR DEVELOPMENT

End