P2L1 Requirements Engineering

Notes 09/08/2014

We will focus on requirements and prototyping.

- *Discuss in depth requirements engineering activities.*
- *Discuss techniques to perform system analysis and design in an object oriented fashion.*

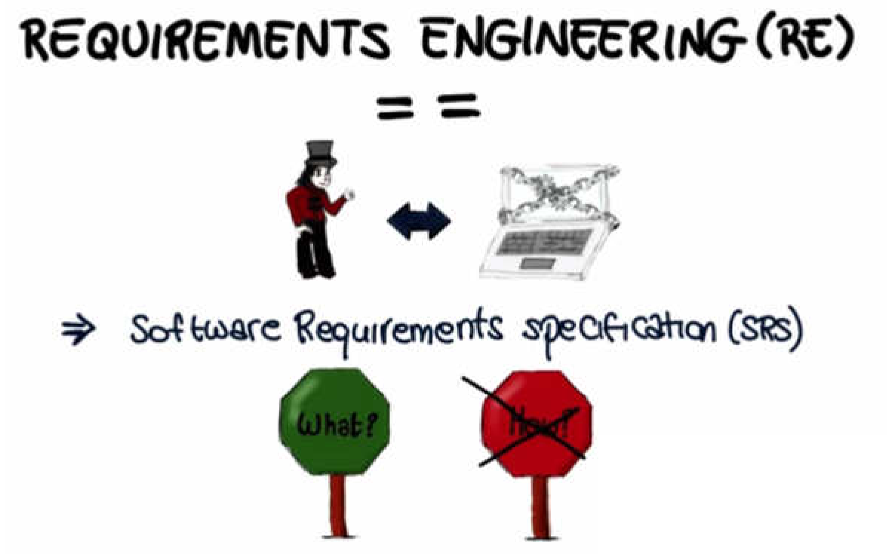
Use engineering techniques to understand and specify the purpose of a software system.

Interview with Jane Cleland-Huang (DePual University) - a world expert in the area of requirements engineering.

- What are software requirements?
  - Software requirements basically provide us a description of what a system has to do. So, typically they describe the functionality of the features that the system has to deliver in order to satisfy its stakeholders. And we usually talk about the requirement specification in terms of what the system's going to do. And we describe it sometimes formally in terms of a set of "shall" statements, that the system shall do this or shall do that. Or we can use various templates to specify those textual requirements. But requirements can also be represented informally in the form of user stories, or use cases, or more formally in the form of state transition diagrams and even in kind of formal specifications, especially for critical parts of safety critical systems.
- What is the requirements engineering?
  - If you notice the term engineering and I'm sure in the other parts of the software engineering process that you're discussing in your course, parts such as testing or coding, don't have the word engineering there and I think one of the reasons requirements engineering has that term is because it covers a number of different activities. So it includes things such as working with stakeholders to elicit or to proactively discover what their requirements of the system are. Analyzing those requirements so that we understand the tradeoffs. So you might have different stakeholders that care about different things, and it might not be possible to deliver all of those things, so we have to analyze the feasibility of the requirements, explore the tradeoffs, merge conflicts. And then of course the specification part, which we talked about a little bit already, and the validation. So did we in fact get the requirements right? Did we build a system that actually matches our requirements? And then on into the requirements management process. And the requirements management process goes through things like

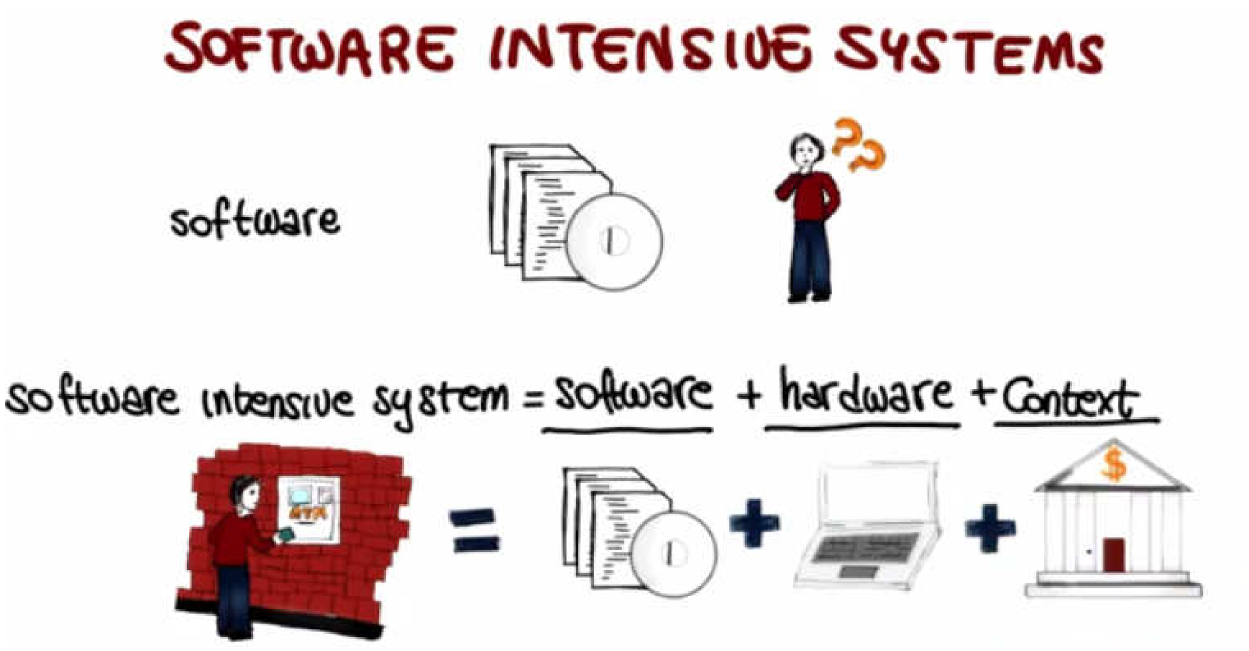
change management. So what if customer or stakeholders need the system to change? How do we manage changing requirements? And I think this is one of the reasons that we've coined the term engineering because it has to be a systematic process which extends across the whole of this is life cycle.

- Why is requirements engineering so important? What happens if we don't do it right?
    - Well, I'm sure that, you know, many people have probably read the kind of report like Standish Report, and other reports of failed project, and things like that and are aware. Now one of the major reasons for projects failing is because we didn't get the requirements right in the first place. So if we don't understand the requirements then we're simply going to build the wrong system. Getting requirements right includes all sorts of things such as finding the right group of stakeholders so we don't exclude major groups of stakeholders. Understanding the requirements correctly. There will be many, many different examples of projects that have failed. For example, in America the healthcare.gov failure, and while we cannot put the blame squarely in the area of requirements, because obviously the project was challenged for a number of different reasons, but clearly it underperformed in many respects related to security, performance, and reliability and these are all parts of the requirements process: things that should have been discovered and the system should have been built in order to meet those requirements. Getting the requirements right in the first place puts us, a project, on the right foot. And so that gives us a much better chance of delivering to the customer what they need. And designing a solution that really meets those requirements. So, it's a critical part of the overall software engineering success.


REQUIREMENTS ENGINEERING (RE)
==
⇒ Software Requirements specification (SRS)
What?
~~How?~~

General RE Definition

- Requirements engineering (RE) is the process of establishing the services that the customer requires from the software system.
- In addition to that, requirements engineering also has to do with the constraints under which the system operates and is developed.
- Very important activity for several reasons.
    - Many errors are made in requirement specifications because we don't do requirements engineering in the right way and many of these errors are not being detected early. But they could be if we were to do RE in the right way.
    - No detecting these errors can dramatically increase software costs.
    - So that's the reason why requirements engineering is important, and why it is important to do it in the right way.
- The final result of the requirements engineering process is a software requirements specification that we also called SRS.
    - Software requirements specification and the requirements engineering, in general, should focus on what the proposed system is intended to do, and not on the how it will do it.
    - How the system will do what it is required to do is something that we will discuss when we talk about design of a system in later phases.


SOFTWARE INTENSIVE SYSTEMS
software
software intensive system = software + hardware + Context
$

Software Intensive Systems

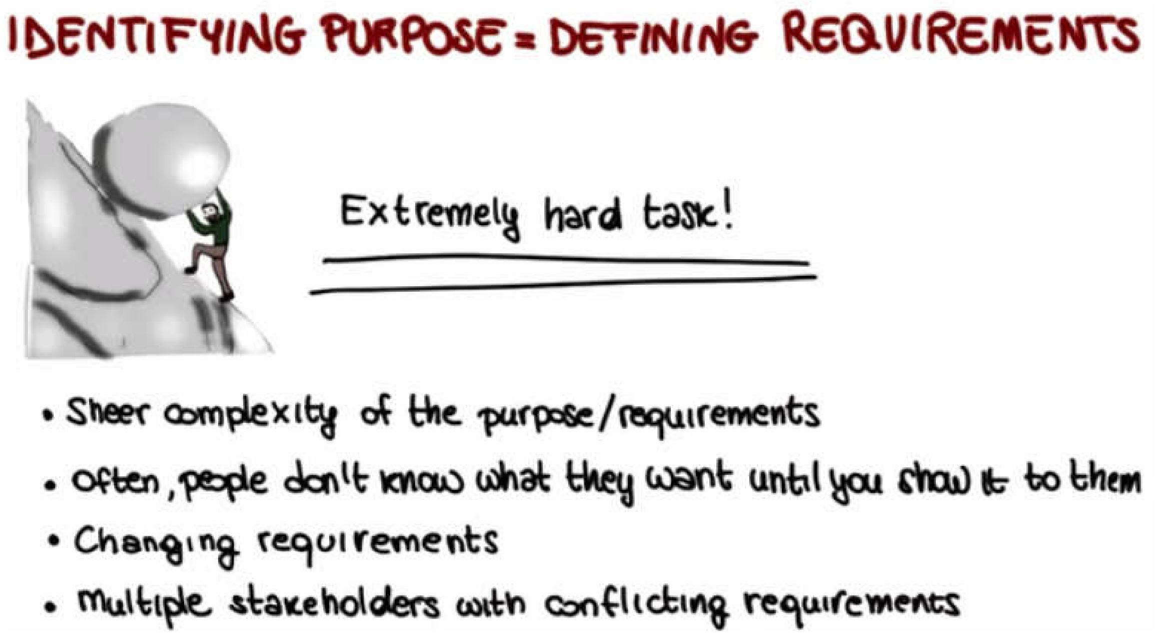
- Software is an abstract description of a set of computations that becomes concrete, and therefore useful, only when we run the software on some hardware, and that, in the context of some human activity that it can support.
- When we say software, what we really mean is a software intensive system.
- It's the combination of 3 things
  - The software
  - The hardware on which the software runs
  - The context in which the software is used.
- Example: A customer, a user, is accessing an ATM machine. And this action involves several things.
  - There is the software that drives the logic of the ATM machine.
  - There is the hardware on which the software runs.
  - There is the context In which the software is used (in this case the bank).
- We usually take hardware and context for granted in this equation, but they actually need to be explicitly considered when building a system.
  - Otherwise, we might forget this is all the functionality, and ultimately of the requirements and we might end up with the wrong system.

SOFTWARE QUALITY
software runs on some hardware and is developed for a purpose
that is related to human activities
Quality = f(,)
FITNESS FOR PURPOSE
Requirements engineering is mostly about identifying the purpose

Software Quality

- Software runs on some hardware and is developed for a purpose that is related to human activities.

- Software quality is not just a function of the software; the software itself does not define the quality of the overall system.
- Software quality is a function of both the software and its purpose, where purpose has to do with the way in which the software will be used.
- So a software system can be of low quality not only because it does not work well (i.e. not only because it crashes)
- Just as importantly, a software can also be of low quality because it does not fulfill its purpose
  - This happens quite often.
  - Not rare for the software producers to have an inadequate understanding, or even a complete misunderstanding of the purpose of the software, of what the users want to do and will do with it.
- We can therefore define the quality of software in terms of fitness for purpose.
  - The more the software fulfills its purpose, the more the software is on target, the higher is its quality.
  - Identifying the purpose of the software, so hitting this target, is exactly the goal of requirements engineering.
  - It is the reason why requirements engineering is such a fundamental activity in the context of software engineering.



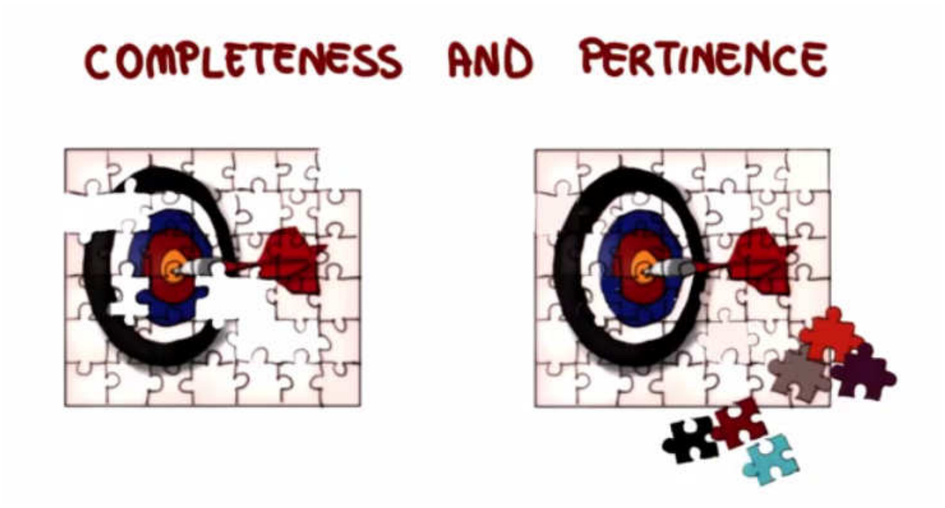
IDENTIFYING PURPOSE = DEFINING REQUIREMENTS

Extremely hard task!

- Sheer complexity of the purpose/requirements
- Often, people don't know what they want until you show it to them
- Changing requirements
- Multiple stakeholders with conflicting requirements

Identifying Purpose

- Identifying the purpose of a softer system means defining the requirements for the system.
- It is an extremely hard task.
- Why is it so hard?
    - The purpose of most systems is inherently, extremely complex, so this has to do with the sheer complexity of the purpose of the requirements.
    - It is very hard to extract from humans this purpose and make it explicit.
        - Steve Jobs: "Often people don't know what they want until you show it to them."
        - It's hard to figure out what people really want.
    - Requirements often change over time.
        - Customers change their mind.
        - Designing and building a system raises new requirements.
        - So for many reasons requirements tend not to be stable, tend to evolve, and that makes it harder to collect them.
    - There are many stakeholders and they often have conflicting goals and requirements.
        - It can be very hard to reconcile the possibly conflicting requirements that might emerge in these cases.
- For all these reasons, it is very, very difficult to perform requirements engineering in an effective way.



Completeness and Pertinence

- These issues and difficulties can result in requirements that show various problems.

- Two particularly relevant and common problems are the lack of completeness and pertinence.
- Completeness refers to the fact that it is often extremely difficult to identify all of the requirements.
    - It is very difficult to have a complete picture of the purpose of the software.
    - What happens is that incomplete requirements are collected and the software is missing functionality that is important for the user.
- Pertinence conversely has to do with the relevance of the requirements.
    - To avoid completeness problems developers often end up collecting a lot of irrelevant when not conflicting requirements.
    - The software could either end up being bloated (might contain unneeded functionality) by these extra requirements or it might even be impossible to build the software due to the conflicting additional requirements.
    - And to make things even worse collecting all of these requirements sometimes doesn't even solve the completeness issue.
        - So you might end up with a set of requirements that is not only incomplete but it also contains extra information that can be harmful to the system.
- The bottom line is that gathering an adequate, accurate, complete, and pertinent set of requirements that identify the purpose of a software system is an arduous task.

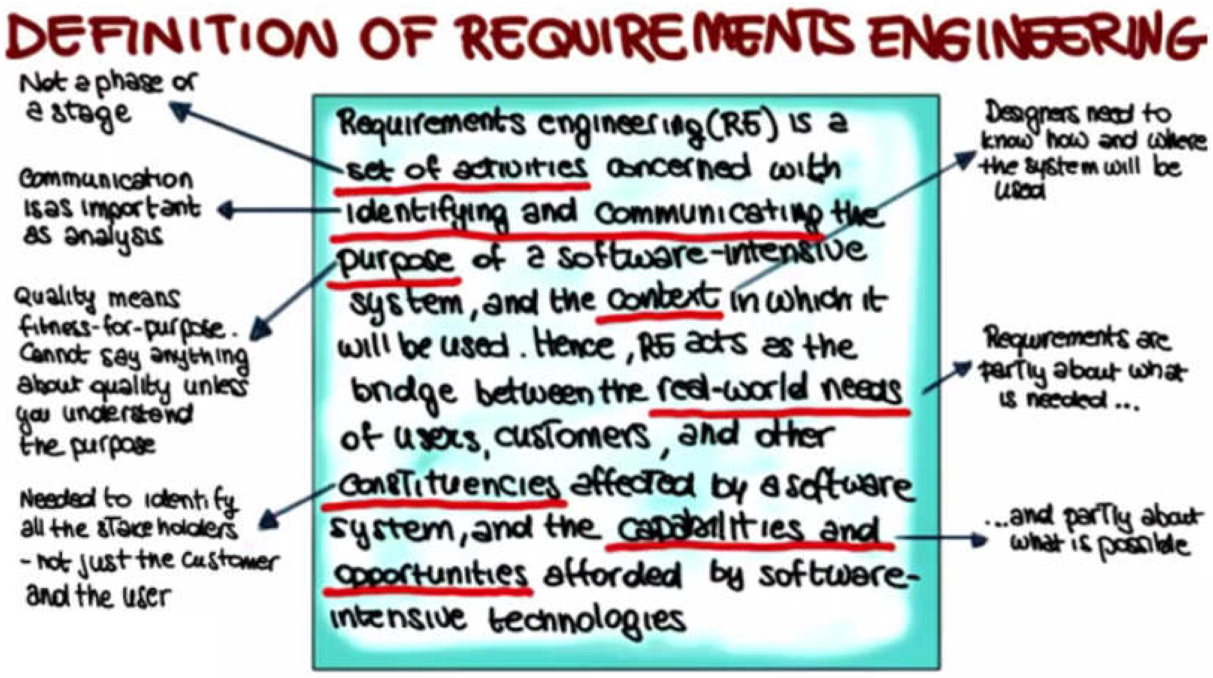
Why can irrelevant requirements be harmful? Why is that a problem to have irrelevant requirements?

- Irrelevant requirements can introduce inconsistencies.
- So they could be irrelevant requirements that not only are not pertinent but they are inconsistent with some of the pertinent requirements.
- They can also waste project resources, because if we spend time designing and then implementing the parts of the system that we referred to this irrelevant requirements, of course, we are wasting project resources.

BEST PRACTICE ?
TIC TAC TIC TAC
BLA BLA BLA
BLA BLA

Best Practices

- How do people cope with these difficulties?
- In practice, developers or analysts usually identify a whole bunch of requirements, sometimes the easiest and most obvious ones.
- They bring those to the stakeholders, and the stakeholders have to read the requirements, understand them, and if they agree, sign off on them.
- The problem is that in general, these requirements documents are difficult to read.
  - They are long.
  - They are often unstructured.
  - They typically contain a lot of information.
  - In general, they are not exactly a pleasant read.
- So what happens is that often the stakeholders are short on time, overwhelmed by the amount of information they're given, and so they give in to the pressure and sign.
- This is a bit of a dramatization but it's clear that what we are looking at is not an ideal scenario.
  - Clearly this is not the way to identify the real purpose of a software system to collect good requirements.
  - And since one of the major causes for project failure is the inadequacy of requirements, we should really avoid this kind of scenario.
- We should follow a rigorous and effective requirements engineering process instead.



DEFINITION OF REQUIREMENTS ENGINEERING

Requirements engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the context in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies

Not a phase or a stage

Communication is as important as analysis

Quality means fitness-for-purpose. Cannot say anything about quality unless you understand the purpose

Needed to identify all the stakeholders - not just the customer and the user

Designers need to know how and where the system will be used

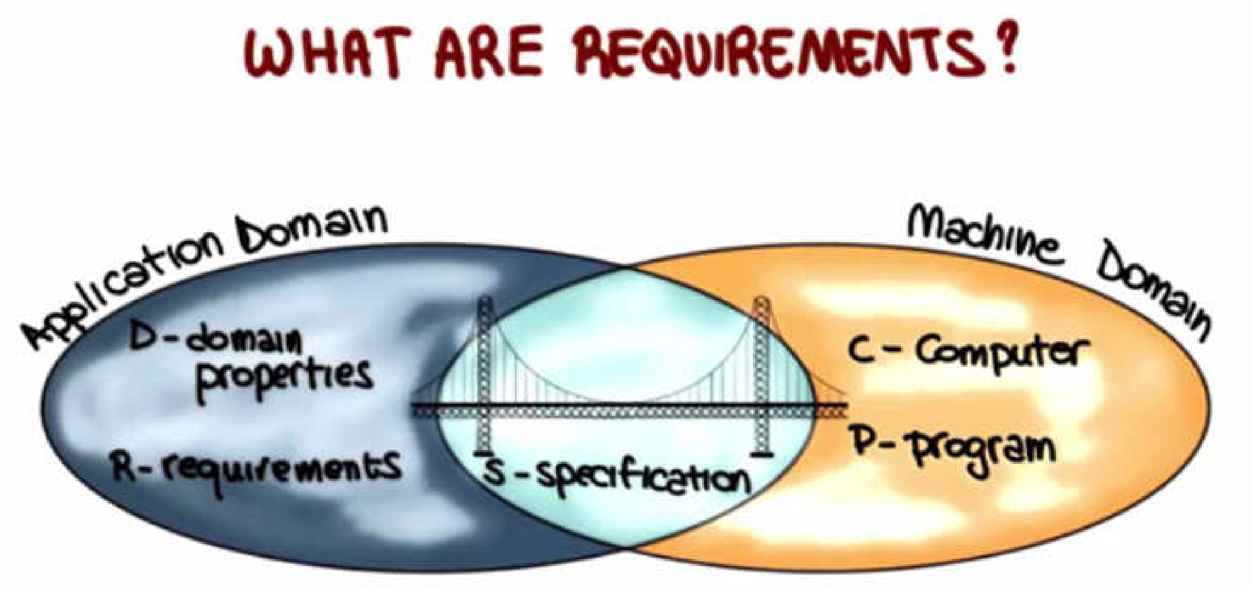
Requirements are partly about what is needed...

...and partly about what is possible

RE Definition Breakdown

- The first part of the definition says, that the requirements engineering is a set of activities concerned with identifying and communicating the purpose of a software intensive system and the context in which it will be used.
- Something we can highlight in here, is the fact that we're talking about a set of activities.
  - So, what that means is that requirements engineering is not just a phase or a stage.
- It also says that it's about identifying and communicating.
  - What that is telling us is that communication is as important as the analysis.
  - It's important to be able to communicate these requirements not only to collect them.
- It explicitly talks about purpose.
  - Quality means fitness-for-purpose.
  - We cannot say anything about quality unless we understand the purpose.
- And the last thing I want to point out in this first part of the definition is the use of the term context.
  - Designers, analysts, need to know how and where the system will be used.
  - Without this information, you cannot really understand what the system should do and you cannot really build the system.

- The second part of the definition says that requirements engineering acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system and the capabilities and opportunities afforded by software-intensive technologies.
- So requirements are partly about what is needed, the real-world needs of all these stakeholders, but they're also partly about what is possible, what we can actually build.
    - We need to compromise between these two things.
- The term constituencies indicates that we need to identify all of the stakeholders, not just the customer and the users
    - So anybody who is affected by a software system.
    - It is very important to consider all of these actors. Otherwise, again, we'll be missing requirements, we'll be missing part of the purpose of the system, and we will build a suboptimal system.

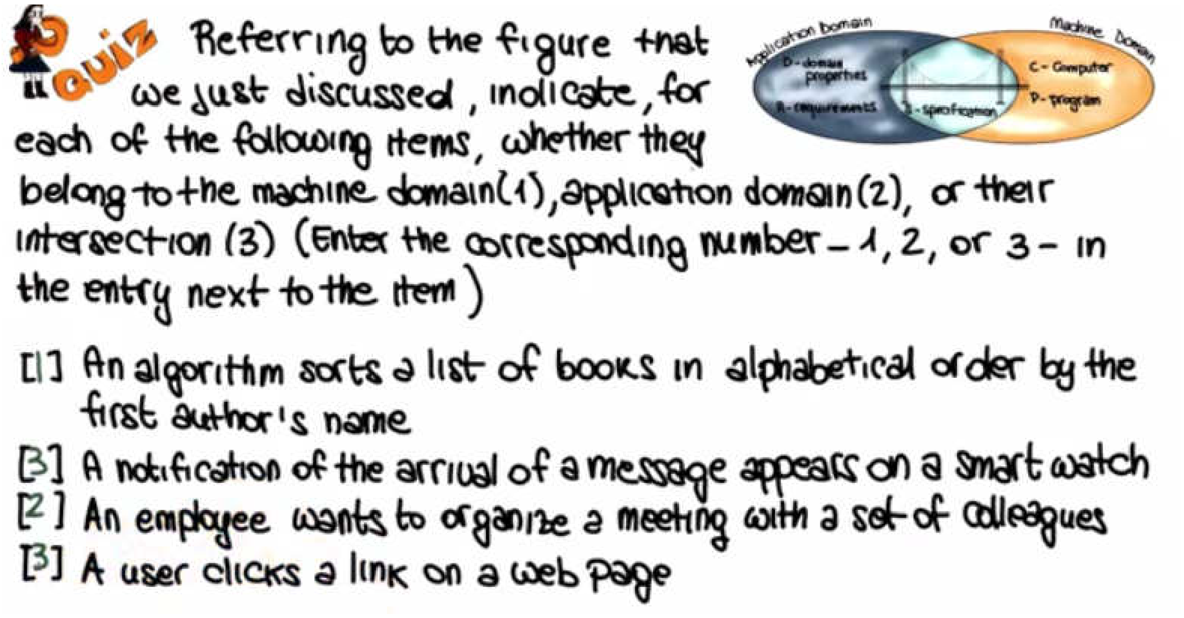


Defining Requirements

- So what is a requirement?
- To define that we'll use this diagram which is a classical one.
- At a high level this diagram contains two main parts
    - The domain of the machine, which is the hardware, operating system, libraries and so on, on which the software will run.
        - And the machine domain is characterized by computers, which are the hardware devices, and programs, which is the software that runs on these devices.

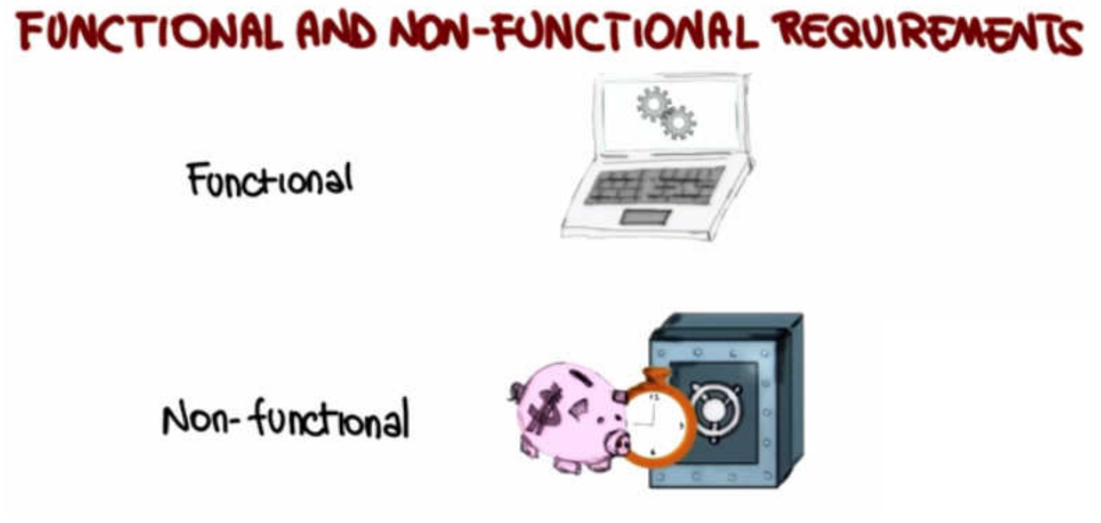
    - The domain of the application, which is a world in which the software will operate.
        - The application domain is characterized by domain properties, which are things that are true of the world anyways, whether I'm building my system or not, and requirements, which are things in the world we would like to achieve by delivering the system that we are building.
        - Basically, to put it in a different way, the former, the domain properties, represents the assumptions that we make on the domain. And the latter, the requirements, are the actual requirements that we aim to collect.
- At the intersection of this application domain and this machine domain is what we normally call the specification, which is a description, often a formal description, of what the system that we are building should do to meet the requirements.
    - This is a bridge between these two domains.
    - The specification is written in terms of shared phenomena.
    - Things that are observable in both the machine domain and the application domain.
- A couple of examples of what these phenomena, these shared phenomena, are.
    - Two main kinds of phenomena.
        - The first one are events in the real world that the machine can directly sense.
            - For example, a button being pushed or a sensor being activated.
            - These are events that happen in the application domain, but that the machine can detect.
            - They're events that can be used to define the specification.
        - And the second type of phenomena are actions in the real world that the machine can directly cause.
            - For example, an image appearing on a screen or a device being turned on and off.
            - This is something that the machine can make happen and then can have manifestation in the real world.
            - And again this is therefore something on which the specification can predicate, something that we can describe in our specification.
- When writing a specification you have to be aware of the fact that you're talking about shared phenomena
    - Events in the real world that the machine can sense and actions in the real world that the machine can cause.

    - So this is what the specification is about, a bridge between these two worlds that define what the system should do to satisfy the requirements.


QUIZ
Application Domain
D - domain properties
R - requirements
S - specification
Machine Domain
C - Computer
P - program

Referring to the figure that we just discussed, indicate, for each of the following items, whether they belong to the machine domain(1), application domain(2), or their intersection (3) (Enter the corresponding number – 1, 2, or 3 – in the entry next to the item)

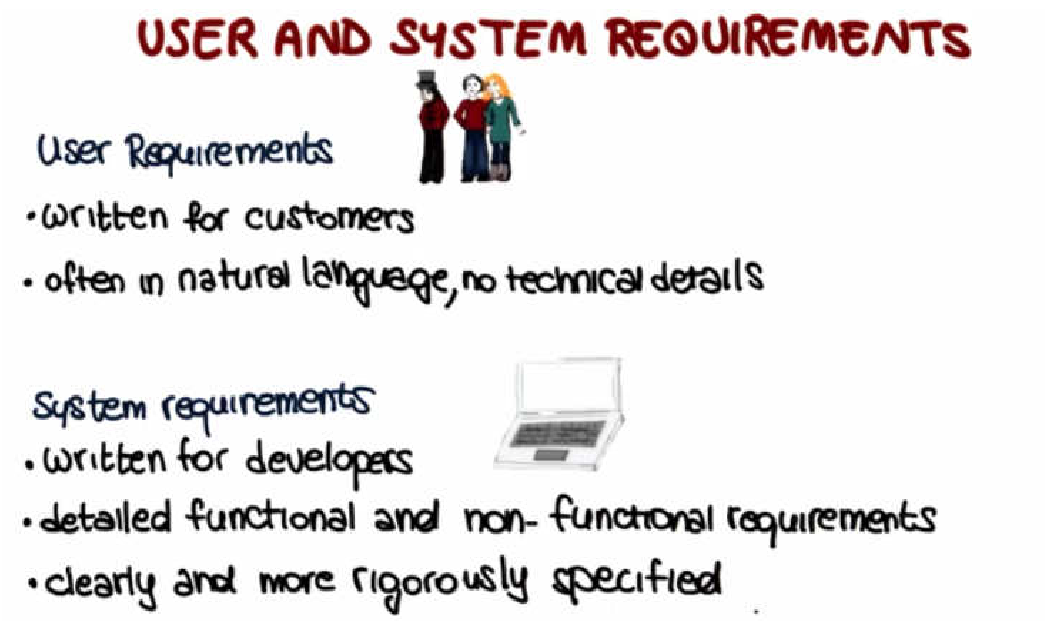
[1] An algorithm sorts a list of books in alphabetical order by the first author's name
[3] A notification of the arrival of a message appears on a smart watch
[2] An employee wants to organize a meeting with a set of colleagues
[3] A user clicks a link on a web page



Functional and Nonfunctional Requirements

- Among the requirement that we can collect from the application domain, we need to distinguish between two main types: functional requirements and non-functional requirements.
- Functional requirements have to do with the functionality of the system, with what the system does with the computation.
    - For example the elevator shall take people to the floor they select.

    - That's a functional requirement, that has to do with the functionality of the system.
    - Or for a very simple one, the system has to output the square root of the number past as an input.
    - In general these kind of requirements have well defined satisfaction criteria.
        - It is pretty clear how to check whether the output is actually the square root of the number passed in input.
- Non-functional requirements refer to a system's non-functional properties, systems qualities such as security, accuracy, performance, cost—or usability, adaptability, interoperability, reusability and so on.
    - These qualities don't necessarily have to do with the functionality.
    - Unlike functional requirements, non functional requirements do not always have clear satisfaction criteria.
    - For example, if we say that the elevator must be fast, that's a non-functional requirement. It has to do with the speed of the elevator, which is a quality of the elevator.
    - But, it, it's not clear how such a requirement could be satisfied.
        - What we need to do in these cases Is that we need to refine these requirements so that they become verifiable.
        - For instance, we might say that the elevator must reach the requested floor in less than 30 seconds from the moment when the floor button is pushed.
        - This is still a non-functional requirement, but is a verifiable one.



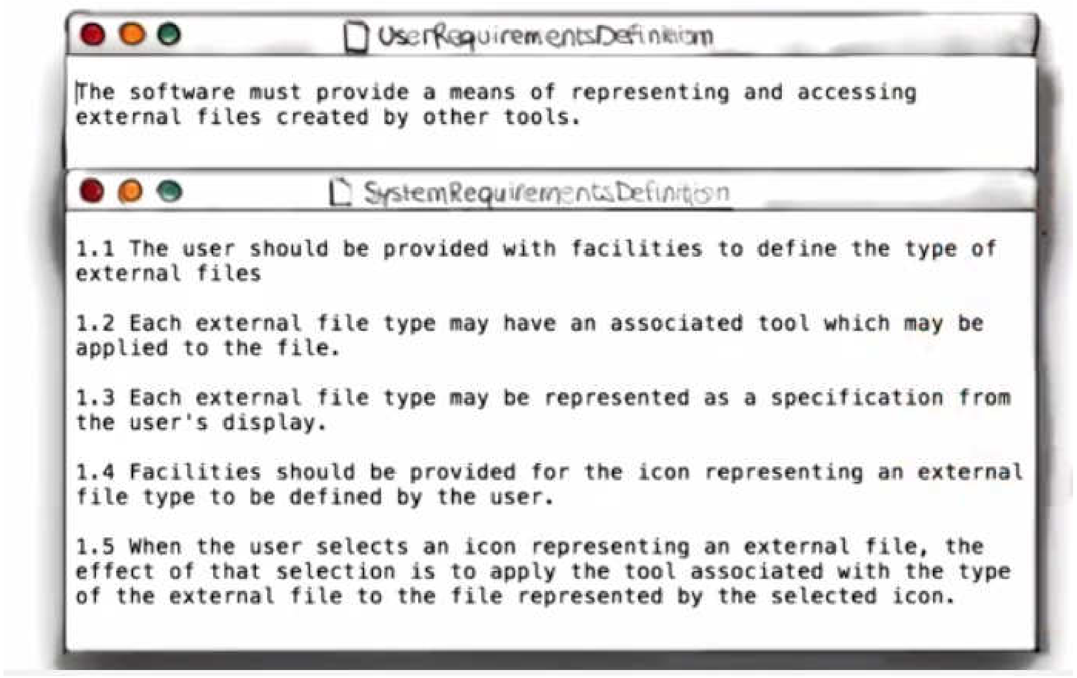
USER AND SYSTEM REQUIREMENTS

User Requirements
- written for customers
- often in natural language, no technical details

System requirements
- written for developers
- detailed functional and non-functional requirements
- clearly and more rigorously specified

User and System Requirements

- Another important distinction, when talking about requirements, is that between user and system requirements.
- User requirements are those requirements that are written for the customers and they're often in natural language and they don't contain technical details.
    - o The reason for that is that their purpose is to allow customers, stakeholders, to check that the system will do what they intended.
    - o So it's a way for the analyst, the developers, to communicate with the customers, with the stakeholders.
- System requirements are written for developers and contain detailed functional and non functional requirements which are clearly and more rigorously specified than the user requirements.
    - o The reason for this difference is that the purpose of the system requirements is to tell developers what to build.
    - o They must contain enough details so the developers can take them and use them to design and then develop a system.



The software must provide a means of representing and accessing external files created by other tools.

1.1 The user should be provided with facilities to define the type of external files

1.2 Each external file type may have an associated tool which may be applied to the file.

1.3 Each external file type may be represented as a specification from the user's display.

1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.

1.5 When the user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

- Above is a concrete example, a user requirement that just says that the software must provide a means of representing and accessing external files created by other tools, and the corresponding system requirement.

- The former is an informal and high level description of a piece of functionality, whereas the latter describes the same functionality but in a much more extensive and rigorous way.
- The latter is something that the developers can use to design and then build a system whereas the former is something that can be used to communicate with the stakeholders, with a non-technical audience.
- We need to define both because they serve different purposes.

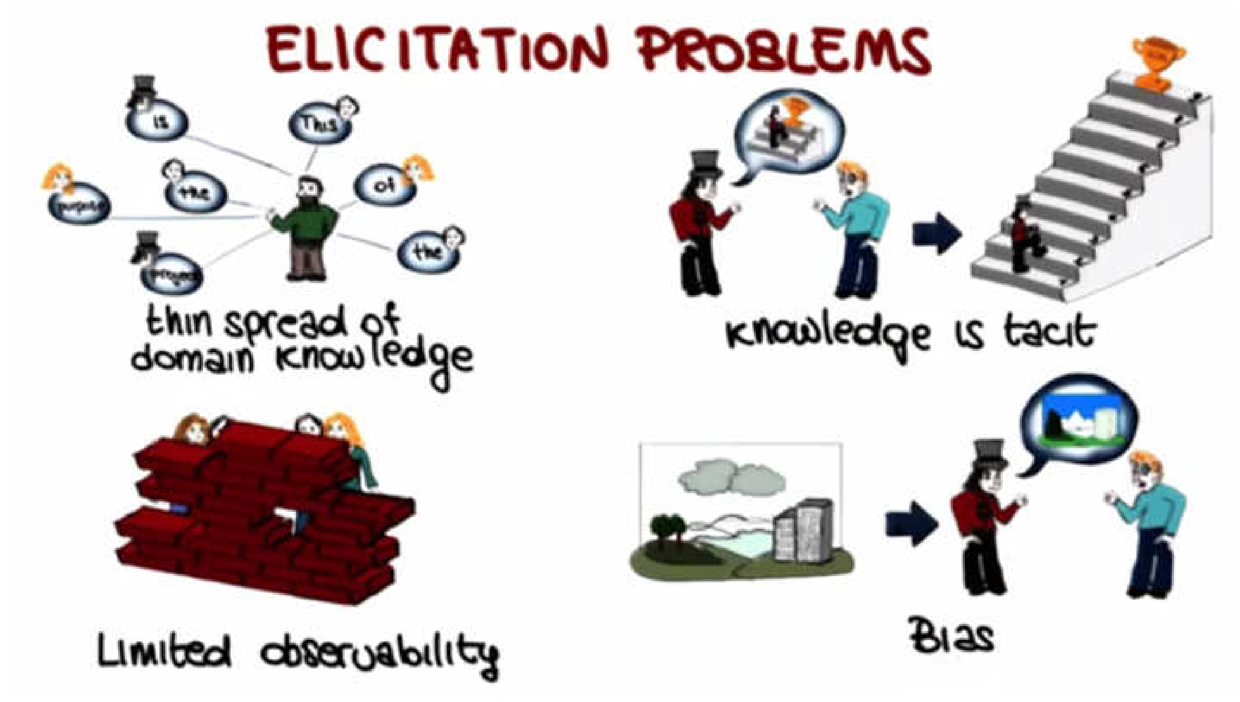
WHERE DO REQUIREMENTS COME FROM?
Stakeholders
...
Application domain
SCHOOL
...
Documentation
CHART
My personal notes
...

Requirement Origins

- Where do requirements come from?
  - Stakeholders: anybody who is effected by the system and its functionality.
    - Customers, users, and so on.
  - The typical social requirement is the application domain.
    - For example, the fact that my software is running within a bank, or within a school.
    - Why is the application domain a social requirement?
      - There are constraints that are characteristics of the application domain that will affect the functionality of the system.
      - For a simple example, just think about regulations, so banking regulations and school regulations in these cases.

- Those are things that might affect the functionality of my system and, therefore, that may become part of my requirements.

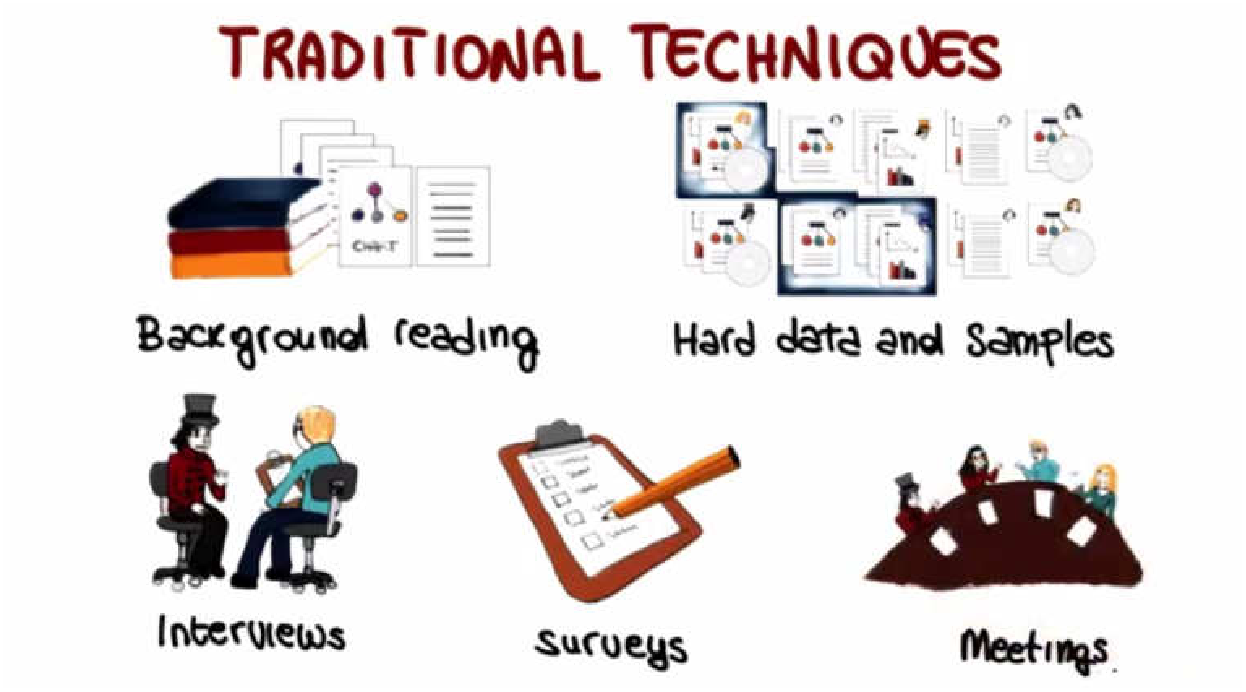
- Documentation can be an additional source of requirements.
  - For example, notes, papers, manuals, books.
  - So everything that refers to the functionality of the system that we're going to build.



Elicitation Problems

- Extracting requirements from these sources is not a straightforward task.
- There are many issues involved with the requirements elicitation.
  - The thin spread of domain knowledge.
    - Knowledge is rarely available in an explicit form, that is, it is almost never written down.
    - Knowledge is often distributed across many sources.
      - To find out the purpose of the project the developer, the analyst, needs to talk to a lot of different people.
    - To make things even worse there are often conflicts between the knowledge gathered from different sources.
  - Knowledge is often tacit, what is also called the say/do problem.

        - For instance we have a customer that is describing to the analyst the way in which he accomplishes a task.
        - So he performs these three steps and reaches the goal.
        - Whereas in practice, the actual way in which this task is accomplished is by going through a larger number of steps to get to the same goal.
        - So even if the knowledge were more concentrated, people simply find it hard to describe knowledge that they regularly use.
        - It is hard to make knowledge explicit, to pass this knowledge to someone else.
    - Yet another problem is limited observability.
        - Identifying requirements through observation is often difficult
            - The problem owners might be too busy to perform the task that we need to observe or they might be doing a lot of other things together with the task that we need to observe, so that it becomes confusing.
            - That introduces noise.
        - The presence of an observer might change their problem.
            - It is very typical for human subjects to improve or modify an aspect of their behavior, which is being experimentally measured in response to the fact that they know that they're being studied.
            - You know that somebody's studying you and you change the way in which you behave. A typical issue.
    - Finally, the information that we collect might be biased for several reasons.
        - People might not feel free to tell you what you need to know.
        - People might not want to tell you what you need to know.
        - In all the common cases in which the outcome might affect them, people might provide you a different picture from the real one in order to influence you.
            - They might have a hidden agenda, and mislead you, either consciously or unconsciously.
- All these issues add to the complexity of collecting requirements, of identifying the purpose of a system.



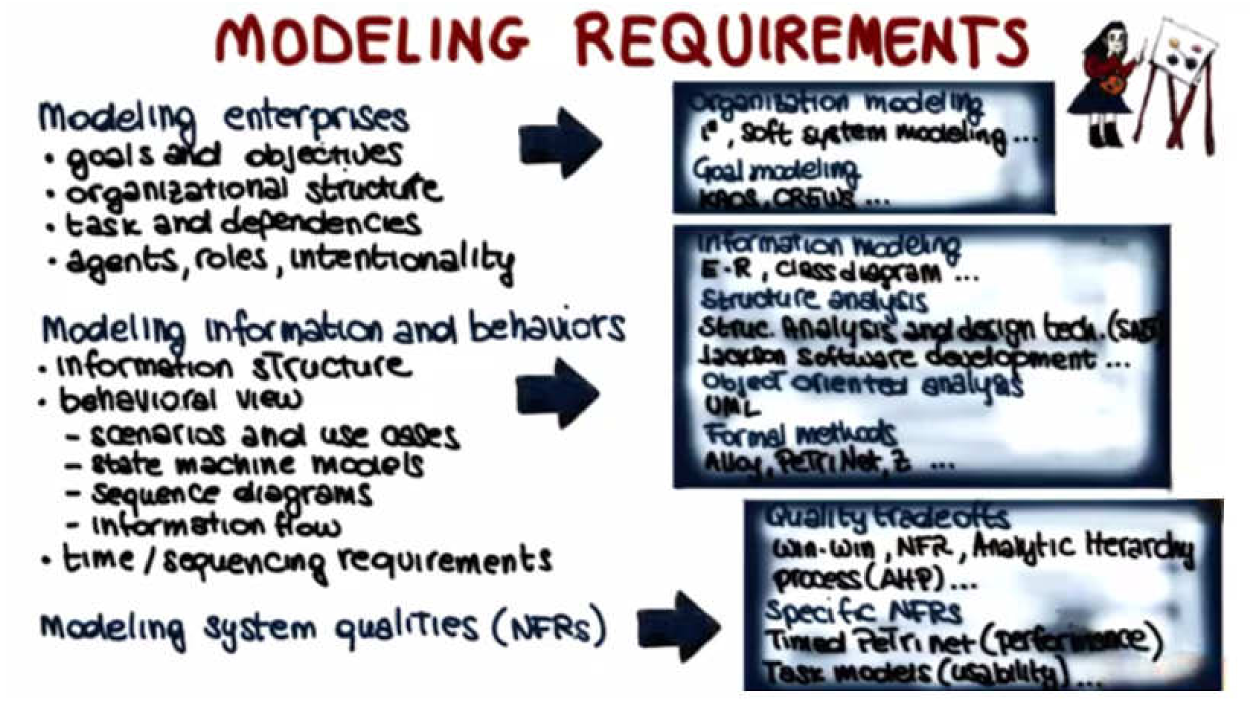
Traditional Techniques

- Many different techniques have been proposed to cover the intrinsic problem of eliciting requirements
- Here we list some of most traditional techniques for requirement elicitation that can be used separately or combined.
  - Background reading –
    - Involves collecting information by reading existing documents such as company reports, organizational charts, policy manuals, job descriptions, documentation of existing systems and so on.
    - This technique is especially appropriate when one Is not familiar with your organization for which the requirements are being collected.
    - One of the main imitations of these kinds of approaches is that recent documents may be out of sync with reality and tend to be long winded.
    - It may contain many relevant details, so you may have to look at a lot of materials to extract enough information.
  - The hard data and samples techniques consist of deciding which hard data we want to collect and choosing the sample of the population for which to collect such data
    - hard data includes facts and figures such as forms, invoices, financial information, server results, marketing data, and so on.
    - The sampling of this data can be done in different ways.

            - The typical way is to do random selection.
    - Interviews are another typical approach for requirement solicitation.
        - Interviews can be structured in which case there is an agenda of very open questions or they can be open ended in which case there is no preset agenda and the interview is more of a conversation.
            - On the positive side
                - interviews can collect a rich set of information because they allow for uncovering opinions as well as hard facts.
                - Moreover, they can probe in depth through follow up questions.
            - On the more negative side
                - Interviewing requires special skills that are difficult to master and require experience and it is not enough to collect a lot of information.
                - If this information is hard to analyze or even irrelevant, it might become useless.
                - So you need to know how to conduct an interview in order to take advantage of these techniques.
    - Surveys can also be extremely useful for gathering new requirements because they can quickly collect information from a large number of people.
        - Moreover, they can be administered remotely by email or through the web.
        - On the other hand, surveys tend to severely constrain the information that the user can provide and might miss opportunities to collect unforeseen, relevant information.
    - Meetings are generally used for summarization of findings and collection of feedback so as to confirm or refute what has been learned.
        - It is fundamental that they have clearly stated objectives and are planned carefully.
        - This is something that should be quite obvious, but doesn't always happen in practice.

Other Techniques

- Some other techniques besides the traditional ones can be divided in three main groups.
- There are collaborative techniques that were created to support incremental development of complex systems with large diverse user populations.
    - An example of such techniques which is widely used and you might know is brainstorming.
- There are also social approaches or techniques that explore the social sciences to better collect information from the stakeholders and the environment.
    - For example, ethnographic techniques are based on the idea of collecting information on the participants by observing them in their original environment.
- Cognitive techniques leverage cognitive science approaches to discover expert knowledge
    - For example, they can be used to understand the problem solving methods.

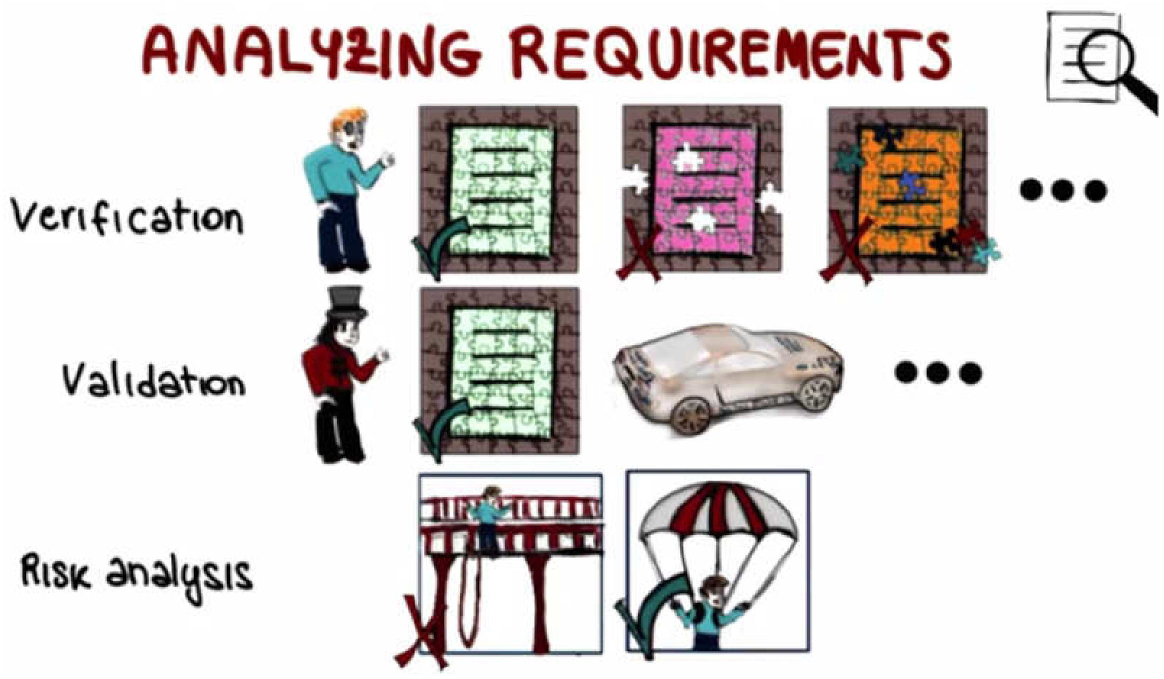


Modeling Requirements

- Once we collected the required knowledge on the requirements for the system that we're developing, we need to model it in a structured and clear way, so that it can be analyzed and refined.
- There are really tons of ways to do this, depending on your focus and objectives.
- More specifically, when modeling requirements you need to decide what you want to model and how you want to model it.
- What you decide to model depends on where your emphasis is, i.e. on which aspects of the requirements you want to focus.
  - For example if your emphasis is on the characteristics of the enterprise of the company that you are analyzing, you may want to model goals and objectives of the company, or its organizational structure, its task and dependencies and so on.
  - Conversely, if your focus is on information and behaviors, you might want to concentrate on aspects such as the structure of information, various behavioral views, or maybe time or sequencing requirements.
  - Finally, if you're mostly interested in the quality aspects of your system, you will focus on the various non-functional properties of the software that are relevant in the context considered.
    - For example, reliability, robustness, security, and so on.

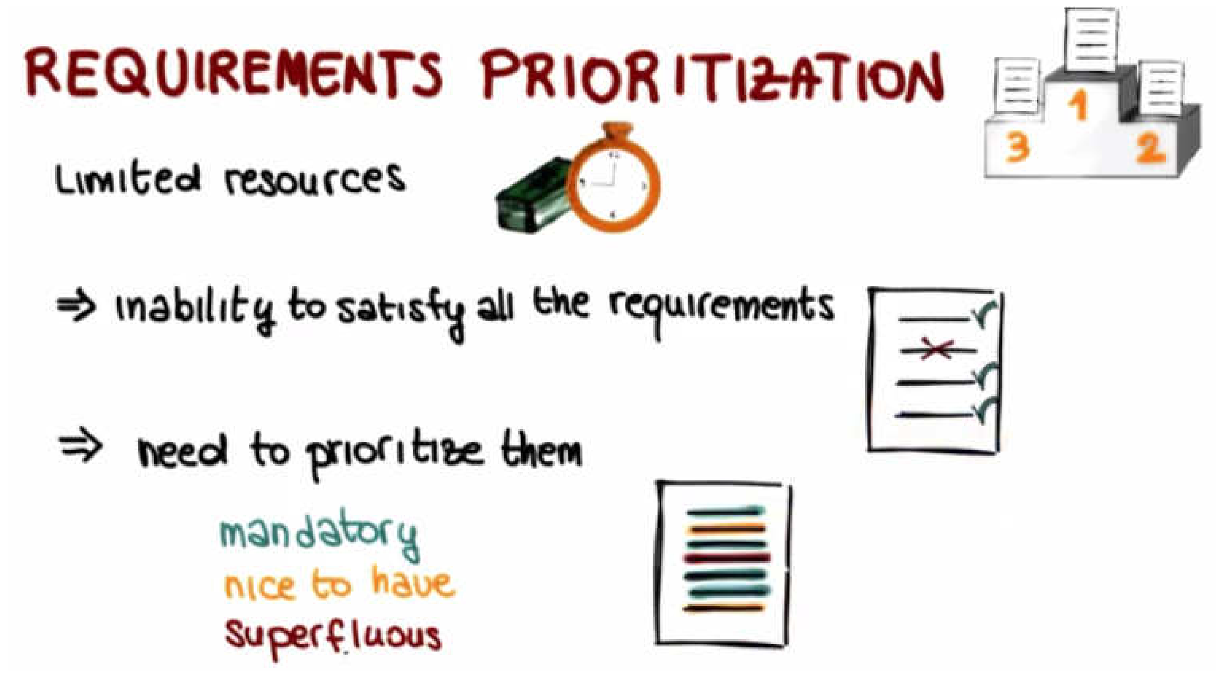
        - You will just pick the ones that are relevant for your context.

- After you have decided what to model in your system, you have to decide how you want to model it.
- Whether modeling enterprises, information, or quality aspects there are many possible models that we can use to represent it and all these models have advantages and disadvantages, different levels of formality and different focus.
- These models are often orthogonal to one another, especially if we consider models in different categories.
    - So what that means is that they're complimentary rather than mutually exclusive.
    - Different models can be used to provide views of the requirements from different perspectives
- As far as we are concerned in the course we will express requirements using one of two main ways.
    - Using natural language - informal specifications
    - Using UML diagrams - graphical models
- Finally goal models are extremely popular.
    - The main idea with goal models is it start with the main goal of the system and then keep refining it by decomposing it in sub-goals.
    - It's a very natural way of progressing and you continue this refinement until you get to goals that can be operationalized, and represent the basic units of functionality of the system.



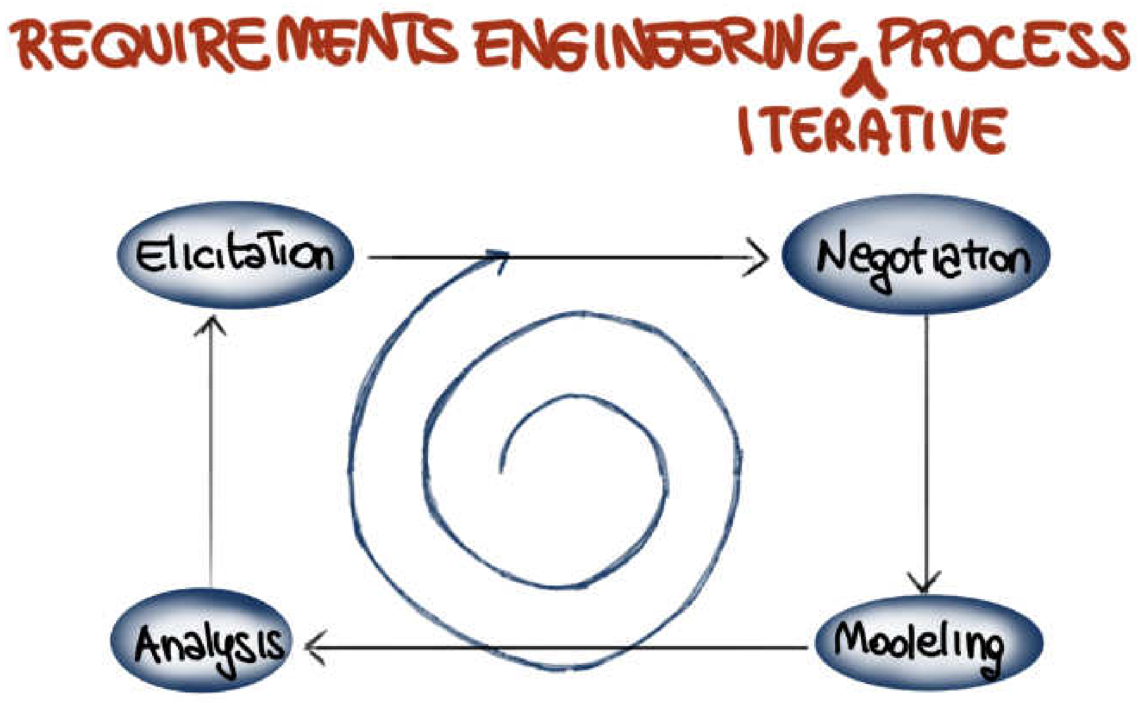
Analyzing Requirements

- Now we are at the point in which we have collected and modeled our requirements.
- The next thing that we can do is to analyze the requirements to identify possible problems
- Specifically there are three types of analysis that we can perform.
  - The first type of analysis is verification.
    - In this case we're talking about the requirements verification.
    - In verification, developers will study the requirements to check whether they're correct, whether they accurately reflect the customer needs as perceived by the developer.
    - Developers can also check the completeness of the requirements, check whether there are any missing pieces in the requirements.
    - They can check whether the requirements are pertinent, or contain irrelevant information.
    - They can also check whether they're consistent, unambiguous, testable, and all those properties that should be satisfied for the requirements.
  - A second type of analysis that is typically performed on requirements is validation.
    - And the goal of validation is to assess whether the collected requirements define the system that the stakeholders really want.
    - The focus here is on the stakeholders and in some cases, stakeholders can check the requirements directly if the requirements are expressed in a notation that they understand or they might check them by discussing them with the developers.
    - Another possibility is that stakeholders asses the requirements by interacting with a prototype of the system, in case the requirements engineering process that is being used involves early prototyping.
    - Finally surveys, testing, and other techniques can also be used to validate requirements.
  - A final type of analysis that we can perform on requirements is risk analysis.
    - Risk analysis aims to identify and analyze the main risks involved with the development of the system being considered.
    - If some requirements are deemed to be too riskythis might result in changes in the requirements model to eliminate or address those risks.
- All of these analysis activities can be performed in many different ways depending on the modeling languages chosen to represent the requirements and on the context.


REQUIREMENTS PRIORITIZATION
Limited resources
⇒ inability to satisfy all the requirements
⇒ need to prioritize them
mandatory
nice to have
superfluous
1
2
3

Requirements Prioritization

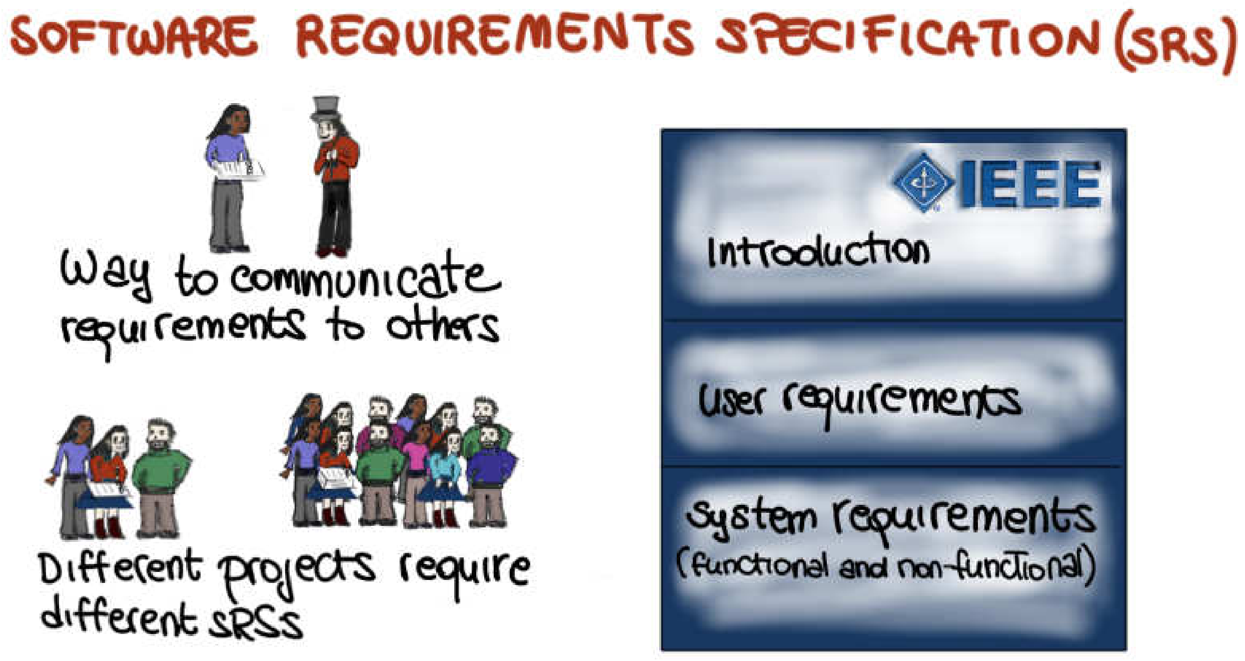
- While collecting, modeling, and analyzing requirements we might realize that the resources available for the project are not enough to satisfy all of them.
  - There's not enough time.
  - Not enough money
  - Not enough manpower.
- Therefore, there are some requirements that we won't be able to satisfy.
- In these cases we must prioritize our requirements by classifying them in one of three classes.
  - The first class is mandatory requirements, and these are the requirements we must satisfy.
  - Then there are the nice to have requirements. They are the ones that we will satisfy if resources allow.
  - And finally, there are the superfluous requirements, and those are the requirements that we're going to keep around, but that we're going to postpone. For example, we might decide to satisfy them in the next release.



Requirements Engineering Process

- Now put together all that we have discussed and see how a requirements engineering process actually works.
- Requirements engineering consists of three main steps.
  - Elicitation of the requirements, in which we extract requirements from various sources.
  - Modeling in which we represent the requirements using one or more notations or formal reasons
  - Analysis, in which we identify possible issues with our requirements
- There is actually a 4th step that we kind of mention but not explicitly.
  - The negotiation that can happen between the stakeholders and the developers during which requirements are discussed and modified until an agreement is reached.
- So if you want to think of this as a process, as a sequence of steps, we can see that we start from elicitation.
  - We start by eliciting initial setup requirements.
  - We negotiate and refine this se.
  - Then we model the resulting requirements.
  - Finally, we analyze such requirements.

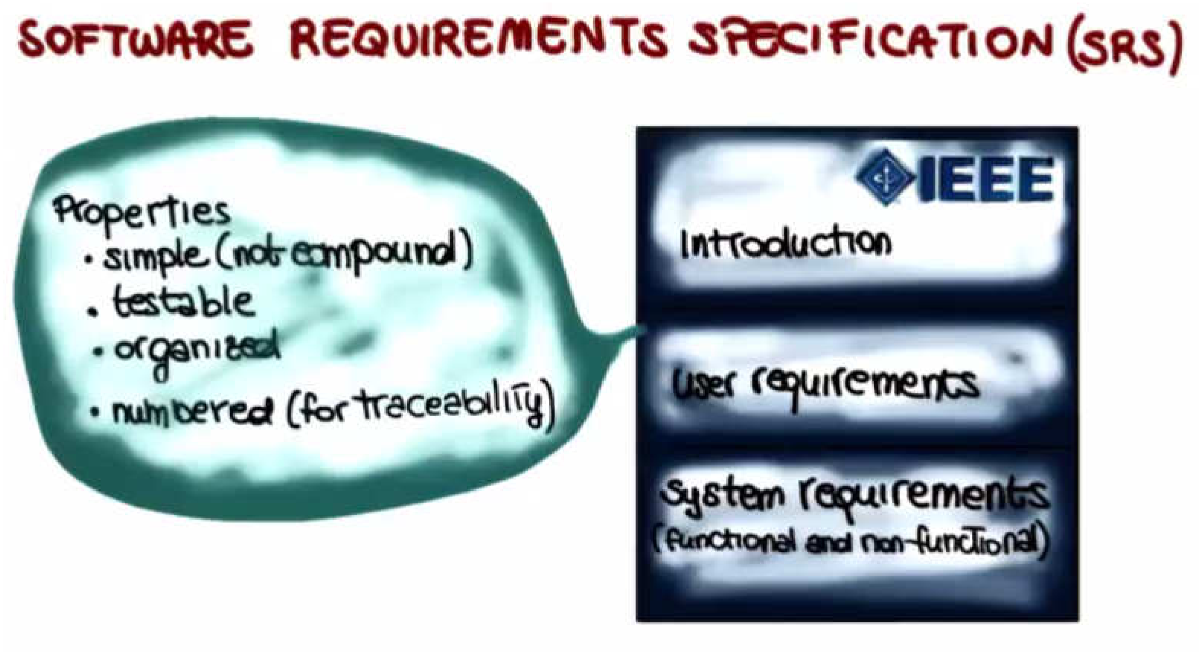
    - However, the process doesn't really stop here because as a result of the analysis, we might have to perform further elicitation.
- So this process is not really a sequential one, but rather an iterative process.
- In practice, we continue to iterate over these four steps gathering a better and better understanding of the requirements at every iteration until we are happy with the settle requirement that we gather and stop the process.


SOFTWARE REQUIREMENTS SPECIFICATION (SRS)
Way to communicate requirements to others
Different projects require different SRSs
IEEE
Introduction
User requirements
System requirements
(functional and non-functional)

SRS - Software Requirement Specification document

- Software Requirement Specification document is an important fundamental way to communicate requirements to others.
- They represent a common ground between analysts and stakeholders.
- Note that different projects might require different software requirement specifications so you need to know your context.
    - For example, the SRS document that you have to create for a small project performed by a few developers can in most cases be a concise and informal one.
    - Conversely the software requirement specification for a multi-project involving a number of developers can be a fairly complex and extensive document.
- So again you have to be aware of your context and build your software requirement specification accordingly.

- In order to have a common format for the SRS document, IEEE defined a standard that divides the document in predefined sections.
- In the context of this course, we will use a simplified version of the IEEE SRS format that includes three main sections.
    - An introduction, which discusses the purpose, context, and objectives of the project.
    - A user requirements definition, which contains the user requirements.
    - And the system requirements specification, which includes both functional and non-functional requirements.


SOFTWARE REQUIREMENTS SPECIFICATION (SRS)
Properties
• simple (not compound)
• testable
• organized
• numbered (for traceability)
IEEE
Introduction
User requirements
System requirements (functional and non-functional)

Recap

- Requirements should be simple, not compound.
- Each requirement should express one specific piece of functionality that the system should provide.
- Requirements should be testable. Untestable requirements such as "the system should be fast" are useless.
- Requirements should be organized.
- Related requirements should be grouped
- More abstract requirements should contain more detailed requirements
- Priorities should be clearly indicated when present.
- Requirements should be numbered, so they can be traced. Numbered requirements allow you to trace them to design implementation and testing elements and items.