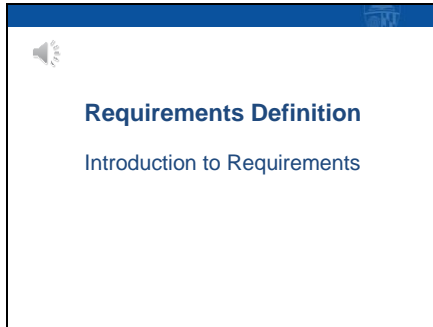


1



In this lecture we'll discuss the different types of requirements.

2



In this lecture I'm going to talk about some very basic but very important things about requirements for software products, and I'm also going to share a few of my several decades of experiences regarding requirements.

I'd like to start off this lecture by asking you to think about what a requirement is. So...before I even get started, I'd like you to pause this lecture, take a minute or two to think about your answer, and write it down. When you're finished, you may resume the lecture.

Before I tell you why I asked you to do that little exercise, I want to share some of my experiences with you. I've been working in the software engineering field for decades...and...I see organizations having the same problems getting good requirements today as I did 30 years ago. Exactly the same problems...and for exactly the same reasons. And it doesn't matter what industry or sector of the economy I've worked with, or whether the organizations were large or small.

So...why have organizations had difficulty getting good requirements for such a long time? There are numerous reasons for this, but I'm going to focus only on one in this lecture: the lack of a common understanding of what a requirement is.

One of the key reasons that this problem has persisted

for so long is very basic...but very important...and is almost always overlooked in organizations. Simply put, different stakeholders on a project have different beliefs about what a requirement is. In my requirements consulting projects I will often get a group of stakeholders together in a room and ask them to write down their definition of what a requirement is. Here's a sample of some of the results I get.

If I have a room full of 10 people, I will get 10 different definitions. If I have a room full of 20 people, I will get 20 different definitions. It never fails.

This is very problematic, because a person's understanding of what a requirement is sets their expectations about things like what kinds of information needs to be gathered, what they think they are responsible for, what they think others are responsible for, and what they think the end result is supposed to be. The bottom line is that most projects start off with what I like to call *uneveled expectations*...and things just get worse from there as the project unfolds.

Doing what I call "leveling expectations" is the first step to getting better requirements. But...how do you do that?

3

What's a Requirement?

REQUIREMENT
A CAPABILITY that must be delivered by the system or a CONDITION that the system must satisfy.

CAPABILITY = Functional Requirement
CONDITION = Non-Functional Requirement

SOURCE: Adapted from IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990, IEEE Computer Society, 1990.

The slide features a blue header with a speaker icon and a photo of a man in the bottom left corner. The main content is a yellow box with the definition of a requirement.

One of the easiest ways to ensure that stakeholder expectations are leveled is to start out with a common definition of what a requirement is and to communicate that definition to key project stakeholders. In this context, key project stakeholders include analysts, customers, end users, developers, testers and managers.

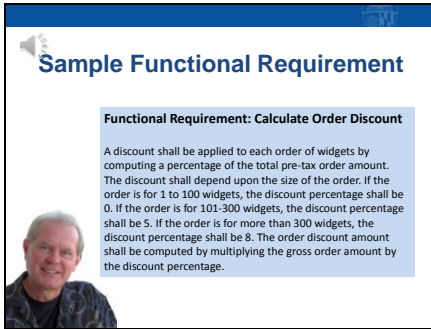
Here's a definition that I've adapted from an industry standard and which I recommend using:
A requirement is a capability that must be delivered by the system, or a condition that the system must satisfy.

I like this definition for several reasons...it's simple, accurate, and easy to remember. And...it maps directly to the two major types of requirements associated with

software development projects...namely, functional requirements and non-functional requirements.

Let's take a look at some examples of each of these requirements types.

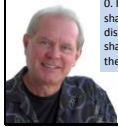
4



Sample Functional Requirement

Functional Requirement: Calculate Order Discount

A discount shall be applied to each order of widgets by computing a percentage of the total pre-tax order amount. The discount shall depend upon the size of the order. If the order is for 1 to 100 widgets, the discount percentage shall be 0. If the order is for 101-300 widgets, the discount percentage shall be 5; if the order is for more than 300 widgets, the discount percentage shall be 8. The order discount amount shall be computed by multiplying the gross order amount by the discount percentage.



Let's start with functional requirements. A functional requirement is something the system must do...in other words, the behaviors the system must possess.

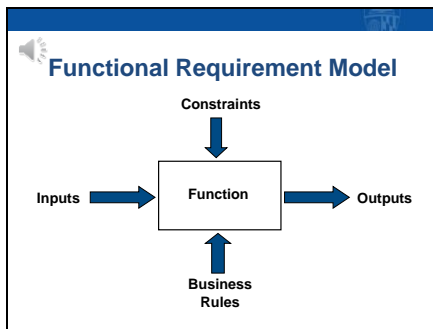
These could be behaviors like calculations, saving information, or displaying information, for example.

Functional requirements can be relatively simple...or...they can be quite complex.

Let's take this example. I'll provide some background information to add context. Let's suppose our company sells widgets, and we are gathering requirements for an order entry system that will assist in processing sale transactions. Our sales department stakeholders need the system to calculate discounts on certain widget orders...so, calculate order discount is an example of a functional requirement...it's something the system must do.

Now, the basic intent of the requirement is easy to understand...but the information required to document the requirement is relatively complex because of the business rules associated with the calculation. This is a pretty well-written requirement, but its business rules are kind of complex...so there is a risk that the rules could be incomplete, incorrect, or misunderstood by those who must develop software to implement it.

5



Here's a way to model a functional requirement that can be very helpful. A functional requirement can have 5 moving parts...inputs, outputs, statement of intended functionality, business rules, and possibly constraints.

Let me map the prior example to the model to illustrate how it works. I'll start with the statement of intended functionality...which is basically the name of the requirement...calculate order discount. A functional requirement can also have inputs. In my example there is one input variable...an order quantity. A functional requirement can have outputs...in my example the output was the calculated order discount amount. And...a functional requirement can also have business rules. Business rules specify how to take the inputs and convert them to the outputs. In my example, most of the written text was the business rules.

A functional requirement may also have constraints. Constraints are criteria that have to be met in the performing of the function. There were no constraints in my example. In practice, constraints are things like a performance requirement that is mapped to a specific function...for example, a query must be performed within one second...or a constraint can be related to organization policy, regulatory policy, accuracy or precision.

Using a model like this one can help to ensure we gather complete and correct information for each functional

requirement.

6



A non-functional requirement is a condition that the system must comply with. Several examples of non-functional requirement are illustrated here.

Note that these examples don't specify behaviors of the system, but rather conditions or constraints that the system must comply with. Non-functional requirements provide important information to the system designers. For example, if the system must support up to 5,000 simultaneous users...versus a single user...the possible design alternatives are quite different.

Note also that these non-functional requirements are written as relatively simple shall statements. They don't have all the moving parts that functional requirements do.