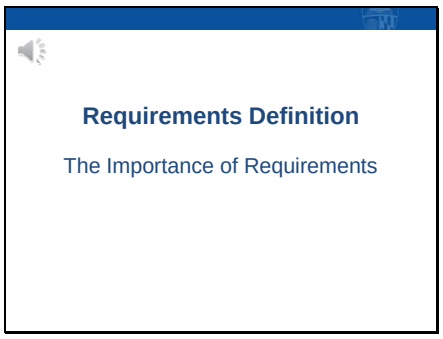
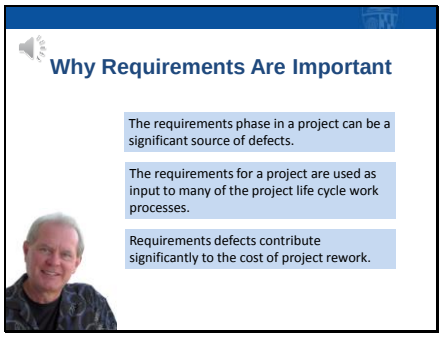
1



In this lecture we will discuss the importance of good requirements.

2



Let's talk about some of the key reasons why requirements are so important.

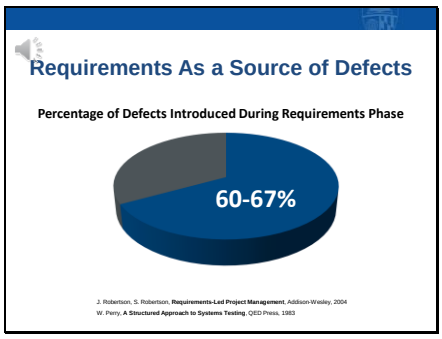
First, the requirements phase in a project can be a major source of defects that get introduced into the project.

Second, the requirements are used as input to many of the subsequent project work phases.

And, third, requirements errors can be costly and time-consuming to fix…and contribute significantly to a project's rework costs.

Let's look into each of these in more detail.

3

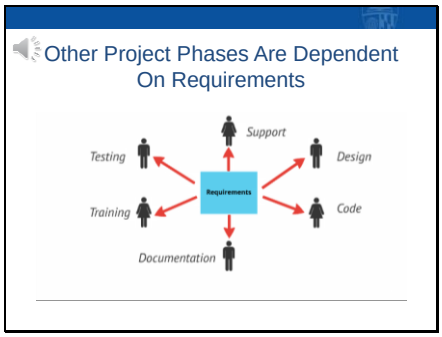


The requirements phase can be a significant source of project errors. More than half of all errors introduced during the course of a project can be introduced in the requirements phase.

That's pretty significant. And…I can tell you from my own experience that I see levels of requirements errors in client projects that are consistent with the statistics indicated on the cited references.

If an organization can improve the requirements process so that these error levels are reduced, there are many cost and schedule benefits to be had.

4

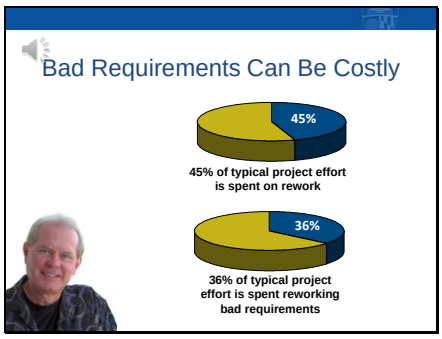


Another reason requirements are so important is that the requirements are used as input to many of the subsequent project work phases.

For example, requirements are used as a direct input to the design process and the test planning process. If the requirements are defective it can have a negative impact on the design and test efforts as well.

And…there can be many other processes that use the requirements as input.

5



Requirements errors can be costly and time-consuming to fix…and there are a number of industry statistics that nicely illustrate the impact of bad requirements in this regard.

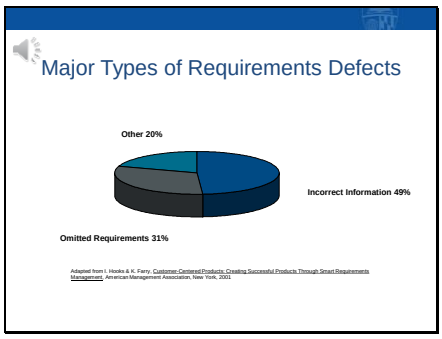
A number of industry studies have measured the percentage of rework effort on projects to range from 40 percent to more than 50 percent of total project effort.

This chart shows a rework statistic that falls near the midpoint of that range…45 percent of total project effort. This indicates that for every 1,000 total person-hours of effort, about 450 person-hours was spent on rework. That's a pretty significant cost impact.

Here's another industry statistic. About 80 percent of rework effort is spent fixing problems that occurred as a result of bad requirements. If we apply this to the first rework statistics…then 80 percent of 45 percent is 36 percent…so, 36 percent of rework effort on a typical project is spent reworking bad requirements. And that's a very significant contribution to project cost…and project schedule.

Imagine if half of the project rework could be eliminated…cost is directly reduced and schedule will also be shortened. In a later course module on software quality, we'll talk more about how bad requirements can influence project cost and schedule.

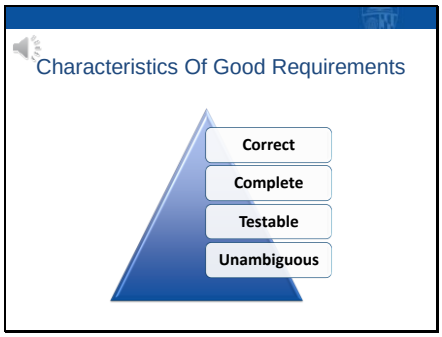
6



If we look at the types of requirements defects that introduced into projects, the results are quite interesting. 80 percent of the requirements defects fall into two categories…incorrect requirements information and omitted requirements information.

So…if we want to reduce the number of requirements defects…one strategy to take is to look at the drivers that cause these two types of errors. So…let's see how we might do that.

7



Let's talk a bit about some of the things that go into making good requirements. In other words, characteristics that are associated with good requirements. I'm going to limit my discussion to the four characteristics that you see here. Good requirements are correct, complete, testable, and unambiguous. I could have added quite a few more characteristics, but these will take us where we need to go in terms of understanding some key things that can be done to help mitigate some major requirements problems.

Note that I put correctness and completeness at the top of the list. That's because 80 percent of requirements problems fall into these two categories as we saw in an earlier slide. Note also that these characteristics are also correlated with each other to a certain extent. For example, an incomplete requirement is also incorrect. A requirement that is untestable may be that way because the requirement is ambiguous, or because the requirement is incomplete, and so forth.

Let's start with correctness. It's pretty obvious that a requirement should contain correct information…so why is incorrectness at the top of the major defects list? Let's look at what may cause a requirement to be incorrect.

One thing that can drive correctness is the identification collaboration effort with key project stakeholders. If the key project stakeholders are not identified, or if they are

not collaborated with…then we risk getting incorrect information. Solution…make sure that our work process includes a stakeholder identification step that identifies all project stakeholders. A simple model for this will be illustrated in the next lecture.

How about completeness? Well, there are two types of completeness…what I call micro-level completeness and macro-level completeness. Micro-level completeness is applied to each individual requirement. Is that requirement completely defined? That's not usually a problem for non-functional requirements, but it's a bigger problem for functional requirements. So…how can this be mitigated? Remember the five-part functional requirements model I illustrated in an earlier lecture…each functional requirement must have a definition, and may have inputs, outputs, business rules, and constraints.

The second type of completeness is macro-level completeness. Macro-level completeness is applied to the entire set of requirements for a project. At that level, the requirements are complete if all the needed requirements for the project have been identified…and that's often hard to assess. So, how can we mitigate this problem? To help ensure that all the non-functional requirements have been identified we can use a checklist of possible non-functional requirements…and simply go through each potential non-functional requirement and determine if it is applicable to our project. Incomplete non-functional requirements are often the result of oversight…so a checklist is a simple tool to deal with this. For assessing the completeness of functional requirements at the macro level, there are two key things that can help: one…using an iterative life cycle, and two…using use cases to capture the project's functional requirements. You'll learn about use cases a bit later on in this course module.

Let me discuss the next two characteristics together since they're highly correlated. A good requirement is unambiguous, meaning it has only one interpretation. A requirement that has multiple interpretations can cause incorrectness as a project progresses. For example, a

software engineer may interpret a requirement in a particular way and base their design and coding assumptions on that interpretation. But, the engineer's interpretation may not match with the business stakeholder interpretation, and it's that interpretation that sets expectations. Ambiguity can be mitigated by choosing an effective documentation technique and by applying the testability test.

A requirement is testable if we can answer "yes" to the following questions: one…can we construct a set of tests to demonstrate whether this requirement has been implemented (or not) in the software product? And, two…whether it has been implemented correctly and completely. If the answer to either question is "no", then the requirement is not testable…and…it may be because it is ambiguous. Let me take an example. Suppose a requirement stated that "all aspects of product performance shall be optimized." That fails the testability test…because it is ambiguous. You can also see how design and coding decisions made by a software engineer's interpretation of that requirement can lead to incorrectness.