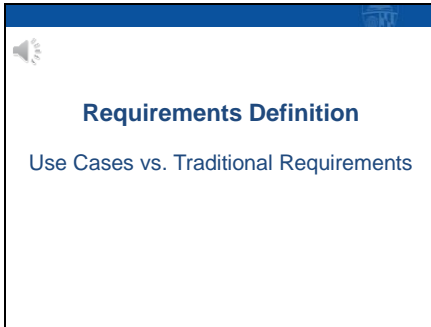
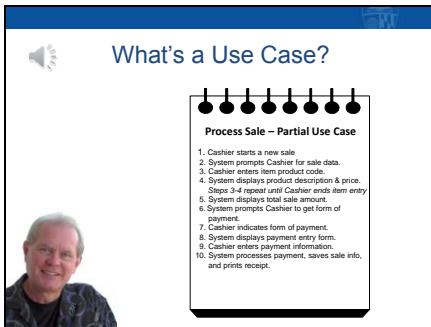


1



In this lecture we'll introduce use cases and discuss how they are used in requirements definition.

2



So...what's a use case? A use case is a way of documenting the functional requirements for a software product.

A use case describes a set of interactions between the product we are gathering requirements for, users of that product, and other systems the product will interact with. Use cases are used to describe the functional requirements for a software product.

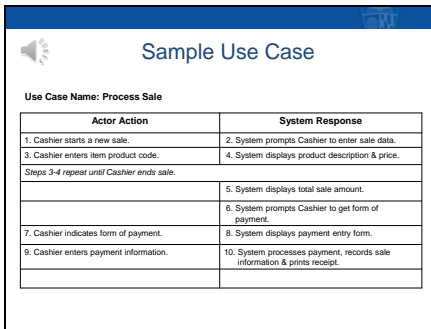
I'll use this sample use case to introduce the concept. First, let me give some background information to provide context. Let's suppose we are gathering requirements for a point of sale software product that will be used by retail stores. One of the things such a system must be able to do is to support the processing of a sale transaction...and a use case like the one shown here might be written to describe the interactions and system functions associated with a sale transaction.

The name of this use case is Process Sale. If you read through the ten steps, you can see that they describe what's involved in processing a sale transaction. Now, this use case is only a partial use case. It's incomplete in a number of ways. For example, the last step involves printing a receipt...but the data elements that must appear on the receipt are not specified yet. Also, the use case doesn't describe the different interactions required if a customer pays by cash, credit card, or check. And it doesn't specify what happens if a customer chooses to

pay by credit card, but the charge can't be authorized. It needs to deal with all of these things, and more, before it's complete. But...it will suffice for our purposes right now.

I'd also like you to notice that a use case tells kind of a story about what needs to be done to process a sale. This "story" nature of use cases is very powerful.

3

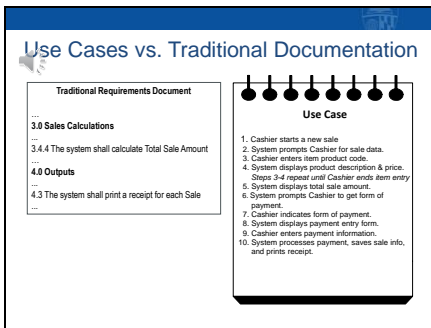


Use Case Name: Process Sale	
Actor Action	System Response
1. Cashier starts a new sale.	2. System prompts Cashier to enter sale data.
3. Cashier enters item product code.	4. System displays product description & price.
Steps 3-4 repeat until Cashier ends sale.	
	5. System displays total sale amount.
	6. System prompts Cashier to get form of payment.
7. Cashier indicates form of payment.	8. System displays payment entry form.
9. Cashier enters payment information.	10. System processes payment, records sale information & prints receipt.

Here's exactly the same process sale use case, but presented in a different format. I call this the two-column format. Things called actor actions are shown in the left-hand column, and system functions are shown in the right hand column. Don't worry about what an actor is at the moment. You'll learn about them shortly.

In practice, the single-column format shown at the beginning of the lecture is the most frequently used format. There's really no difference except for the visual appearance...the same information is presented. In this lecture, I'm going to use the two-column format a lot, only because it makes better use of the video display real estate.

4



Traditional Requirements Document	Use Case
3.0 Sales Calculations 3.4.4 The system shall calculate Total Sale Amount 4.0 Outputs 4.3 The system shall print a receipt for each Sale	1. Cashier starts a new sale 2. System prompts Cashier for sale data. 3. Cashier enters item product code. 4. System displays product description & price. Steps 3-4 repeat until Cashier ends item entry 5. System displays total sale amount. 6. System prompts Cashier to get form of payment. 7. Cashier indicates form of payment. 8. System displays payment entry form. 9. Cashier enters payment information. 10. System processes payment, saves sale info, and prints receipt.

Use cases are a different way of documenting requirements compared to the traditional way of documenting requirements. And...by traditional way I mean without using use cases.

Here's a skeleton of pieces of a traditional requirements document. One way such a document might be organized is to have a section for system inputs, a section for system outputs, one for calculations, and so forth.

Now...let's assume for our sample point of sale product, that the requirements document is 70 pages in length.

And...let's assume that the requirement that the system calculate total sale amount is on page 17 of that document, and the requirement for printing the receipt is on page 55 of that document. As a reader of that document, if I'm interested in learning about all the functions required to process a sale transaction, I need to parse through the document and try to pick out all the related functional requirements...one is on page 17, one is on page 55, and the others will be scattered throughout the document. This presents quite a challenge. If I'm a developer, or a tester, it's really important that I be able to find all the related functionality. If some required functionality was inadvertently left out of the document or not identified during the requirements process...it may be very hard to tell.

Now, let's compare this to the use case approach. Take a look at every step in the use case that uses the word system. Each step that contains the word system represents at least one primitive level functional requirements. The system has to prompt the cashier, display running totals, calculate total sale amount, display a payment entry form, save the sale information and print a receipt. That's at least a half dozen primitive level functions...but they're all related to processing a sale transaction...and they're all documented right here in that single use case instead of being scattered throughout 70 pages of documentation.

This has many advantages. If there's a missing function it will be a lot easier to identify it. Also, the story nature of the use case presents readers of the use case with a vision of the future product. If I'm a tester, the use case will form the basis of my test scenarios and identify all the functions that need to be exercised.

In an earlier lecture we learned that 80 percent of requirements defects are due to incomplete or incorrect requirements. We also learned about macro level requirements completeness...did we identify and document all the needed functionality. Use cases significantly help to ensure macro level completeness compared to the traditional approach.

