|  |  |
| --- | --- |
|  | **Testing Track\_YOL\_QA\_Requirements Management using Use Case** |

Welcome to the session on Requirement Analysis Including- ACTIVITY Diagrams

What will we know at the end of this course:

Overview of Requirements

Details of ACTIVITY diagrams.

Drafting an activity diagram

Deriving test cases using an ACTIVITY diagram

CASE STUDY – for detailed explanation

Let’s have a quick recap of what we learnt earlier

A quick tidbit on Requirement Analysis…

A condition or capability needed by a user to solve a problem or achieve an objective.

A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, objective or other formally imposed document.

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is inevitable as requirements may serve a dual function

May be the basis for a bid for a contract - therefore must be open to interpretation;

May be the basis for the contract itself - therefore must be defined in detail;

Both these statements may be called requirements.

Studies have shown that spending time doing requirements management can pay big dividends in terms of on-time, on-budget project delivery. If a project doesn't get the requirements right, it is difficult to implement correctly or test correctly. Without good requirements management, all aspects of the project will suffer.

“The ability to describe and follow the life of a requirement, in both forward and 21 backward direction (i.e. from its origin, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)”. This implies that traceability allows a requirement to be traced to its initial source as well as to its ultimate implementation throughout the entire life cycle of the project.

In general, a requirement is traceable if it is possible to identify who suggested the requirement, why the requirement exists, what requirements are related to it, and how that requirement relates to other information such as system designs, implementation, and user documentation.

4. Essentially changes in the requirements will need to be tracked/monitored closely. There has to be a detailed analysis of what are the various aspects that these changed requirements could impact on. For instance- there could be additional requirements being added- We will need to identify the additional effort(in terms of time) that will need to be spent. We will also need to identify the impact on the number of modules that these changes may have. There could new risks that could have risen because of these changes- all of which needs top be documented

Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish.

Energy should be directed towards ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements.

Understand the scope and objective of the new system

Define boundaries and know which are all the types of testing which are in scope

Develop the test strategy(consisting of details of the approaches, techniques, the various parties involved, the environment)to validate the system

The above slide explains in details the various sub phases/activities that take place during the Requirement Analysis Phase:

Phase 1: Input:

Get the detailed Business Requirement Documents/ Software Requirements Specifications from the client

These documents can be in any form and it varies from one client/business to another

Have a version control of the documents that you receive- which essentially means track all the different document versions that you receive in a common share path

Phase 2: Process:

Analyze and understand the documents- once the documentation- spend adequate time in understanding the details of the document. It is very essential so that you do not miss out any of the finer details

Review it with an SME or BA- Review the documents that you received by a Subject Matter Expert(SME) or a Business Analyst(BA)

Raise Q&A’s-

Raise all the clarifications at the appropriate time.

Do not have any pending line items as queries.

Track the clarifications to closure-

Maintain a clarifications tracker

Update the tracker with the questions as well as the responses that you receive

Most importantly, refer to these clarifications/responses during the test design activity- so that you lessen unwanted defects

Prepare understanding docs and then have it reviewed by BA's to know whether the understanding is correct.

software requirements specification (SRS): a requirements specification for a software system – is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

Use cases: A use case is a structure for documenting the functional requirements for a system, usually involving software, whether that is new or being changed. Each use case provides a set of scenarios that convey how the system should interact with a human user or another system, to achieve a specific business goal. Use cases typically avoid technical jargon, preferring instead the language of the end-user or domain expert. Use cases are often co-authored by requirements engineers and stakeholders. Use cases are deceptively simple tools for describing the behavior of software or systems. A use case contains a textual description of the ways in which users are intended to work with the software or system. Use cases should not describe internal workings of the system, nor should they explain how that system will be implemented. Instead, they show the steps needed to perform a task.

Prototypes: Prototypes are Mockups of an application, allowing users to visualize an application that has not yet been constructed. Prototypes help people get an idea of what the system will look like, and make it easier for projects to make design decisions without waiting for the system to be built. Major improvements in communication between users and developers were often seen with the introduction of prototypes. Early views of applications led to fewer changes later and hence reduced overall costs considerably. Prototypes can be flat diagrams (often referred to as wireframes) or working applications using synthesized functionality. Wireframes are made in a variety of graphic design documents, and often remove all color from the design (i.e. use a greyscale color palette) in instances where the final software is expected to have graphic design applied to it. This helps to prevent confusion as to whether the prototype represents the final visual look and feel of the application

Details of what is tracked in a Software Requirements Specifications document:

Introduction- An overview of what the project is all about.

Purpose: What purpose is this project being developed for, who are the intended beneficiaries of this project.

Definitions: Description of the various terminologies used in the overall document- this could be the names and the explanation of the various entities in the module. For instance- If we are referring to a Payment Gateway in a shopping Cart module- we could define the various entities involved in that page.

System Overview: A detailed explanation of the overall system to be tested. It’s purpose, the interactions that it would have with the external sources, the various dependency factors

References: Gives us a quick check link for understanding the document in detail

Overall Description: Overall explanation of the project/module to be tested.

Product Perspective: To know on the exact purpose and the benefits of developing and testing such a module

Product Functions: To know the intended/desired output of the module to be tested.

User Characteristics: Details of what the user can do with the product

Constraints, Assumptions and Dependencies: A list of what constraints(of what has not been developed or what is that cannot be tested), assumptions(an understanding based on which the product was developed),dependencies(the dependencies that this product may have and that will also need to be tested when the product is being tested as a whole)

Specific Requirements: Requirements that will need to be focused upon.

External Interface Requirements: Details of external interfaces that the project may have

Functional Requirements: This is the most important area that testers should focus on- as this will give us the detailed idea of what is to be tested and what can be left out.

Performance Requirements: Hardware/Platform details indicating the details of basic requirements for the product to function on expected lines

Design Constraints: Details of what could not be tested

Logical Database Requirement: Details of the set up required to test the product

Software System Attributes: Details of the software configuration of the system being developed/tested

Other Requirements: Miscellaneous details

A requirement Analysis Document consists of :

Introduction

Current system

Proposed system

Overview

Functional requirements

Nonfunctional requirements

Constraints (“Pseudo requirements”)

System models

Scenarios

Use case model

Object model

Data dictionary

Class diagrams

Dynamic models

User interface

Changed Requirements and Impact Analysis

Glossary

Typical functional requirements are:

Authentication

Authorization –functions user is delegated to perform

Typical Non Functional Requirement’s

Static Volumetric

Scalability

Capacity

Reliability

Recoverability

Maintainability

Serviceability

Security

Regulatory

Manageability

Environmental

Data Integrity

Usability

Interoperability

This slide details about the functional requiements.

This slide details about the non functional requirements.

We have all been part of projects where developers say things like “When will user supply us with the requirements that we need. That way we can do what we like to do, code.” In the real world we have never seen this happen. In fact with experience we have found that this is undesirable. Rather a successful project will find a way for both users, developers and test engineers to work together to elicit and manage requirements.

This slide talks in detail about a very important aspect in the Requirement Analysis Phase. Requirements Traceability defines a relationship between two or more requirements.

It ensures that each and every requirement is identified in such a way that it can be associated with all parts of the system where it is used.

Components of RTM:

Test Requirement

Test Scenarios

Test Cases

Test Logs/Defect Ids

The following slides will help us understand the next steps involved once we get a good grip of the requirements

We have now divided the entire Requirement Analysis phase and its follow up of activities in terms of a three structure.

Input: The input to this overall process is the detailed requirements received from the client. We will first need to understand the overall requirements based on various modules involved

Process: We will then need to break down these requirements to represent both the business as well as operational workflows of the system being tested

Output: The overall output of this phase will be the granular version of the requirements. An activity diagram or a workflow diagram depicting the requirements so that we have a very clear picture of what is to be done and how the various modules of the system being tested interact with each other

Activity diagrams represent the business and operational workflows of a system. It is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

The first step is to analyze the requirements and convert the same into an Activity Diagram. To do this, break down the requirement into individual tasks such as processes (login), decisions etc and connect them. The activity diagram should be a pictorial model of the system.

The upcoming slides will have the details of these techniques-PATH Analysis and Model Based Testing

Path Analysis: Path analysis is a process specially developed for turning activity diagrams into test cases. Test flows can be executed in many possible ways and each thread of execution is called a path. Each test case will be derived from a basic path, alternate path, and exception path. Path analysis also helps us reducing the number of test cases and validation of the scenario. It also simplifies the coupling of test cases and test data.

Our approach uses activity diagrams as the model and path analysis technique to automatically generate and maintain test cases. Testers make models of the AUT in their minds. An Activity Diagram is a logical visual representation of this model. Using models in development has been in vogue for quite some time. Models depict the system under study and models have always been a powerful method of capturing and depicting requirements since they are the natural extension of requirements. Some of the models are textual while others are pictorial. A visual representation is easier to analyze compared to a textual document. Hence activities such as reviews and knowledge transition become faster, easier and more effective.

cases and test data.

Test Case Generation using MBT: MBT(Model Based Testing) uses 3 step processes for generating test cases .

Draw Activity Diagram: Typically the requirements are shared in many forms: Functional Requirement Specifications, Project Requirements Document, Use cases, etc. The same is modeled in the form of an activity diagram. We can also generate activity diagrams by reverse engineering the code for test case generation and execution, if it is an existing system.

Determine all possible Paths: A path is a thread of execution through the activity diagram. The ground rule used for determining the path is very simple – Beginning from the Start each time, list all possible ways to reach the end, keeping the direction of the flow in mind.

Generating Test Cases: Using path analysis MBT automatically generates all possible unique test cases and enables the user to assign priority and criticality of the test cases. Once the Activity Diagram is drawn, ADPART automatically generates test scenarios, allows the tester to attach criticality to the scenarios and generates test cases. The scenarios generated are exhaustive and the time taken is a tiny fraction of the time taken by the tester to identify a non-exhaustive set of scenarios. The scenarios can then be categorized in ADPART 2.0 based on criticality. ADPART 2.0 allows the entry of test steps and test data making the test step input a onetime activity, i.e. the tester will never have to input the same test step more than once for the whole set of test cases in a module (area of application modeled in one activity diagram). Once the test steps are input the test cases are generated in an excel spreadsheet instantaneously at the click of a button. The test cases thus generated are in a format compatible with popular test management tools like Quality Center.

The activity diagram will have to be modeled as a stepwise logical flow through the application. This means that an action state should be followed up with the next logical action.

Actions can be user actions like clicking a button or system responses like display of a certain screen

There are many advantages of using Activity Diagrams to model the application. For instance:

Pictorial representation enhances design quality and facilitates review and knowledge transition activities

Significant decrease in time spent in test case design and authoring

Assists in improving test coverage Effort reduction through automation of repeatable tasks Smart test case optimization and test suite creation

Accurate impact analysis for regression testing Easy test case maintenance - Functional evolutions are easy to introduce in test iterations Generated artifacts that are compatible with commonly used test management tools

Challenges:

Drafting/Designing of the ACTIVITY Diagrams can be time consuming.

There could be instances of these diagrams getting exploded- leading to generation of large number of test cases

Risks:

Proper understanding of the requirements is very important

Error in representation of ACTVITY diagrams will lead to erroneous test cases

Considering another example, if suppose we want to install some tool through a wizard, what we generally do is we choose relevant options in each screen and keep clicking next and we finally click finish. This process involves a step by step procedure ,this can be represented as an activity diagram by depicting each action of choosing options and clicking a button as different nodes in the diagram.

For eg: If we have a requirement where it says user should be able to login with a valid password and homepage should be displayed and if user enters an invalid password then an error message should be displayed, the activity diagram will look like the below.

Based on path analysis technique, the above diagram will generate two test cases, one for a valid scenario and another for an invalid scenario (as it has two paths, flow traversing from start node to end node).

Cognizant Testing Services’ Testing Internal Innovations (TII) group has applied the model based approach to capture requirements effectively and generate end – 2 – end test scenarios / cases for flawless coverage.

Our approach uses activity diagrams as the model and path analysis technique to automatically generate and maintain test cases.

MBT also produces node based estimates, coverage metrics and assists in effective regression testing.

In addition, team can organize / build suites like risk based test suites, sanity and negative test cases for executing them at different point of execution life cycle.

ADPART is an innovative, tester-friendly tool. ADPART is targeted at test case design, test case authoring and regression testing. In addition to these features ,it enables you to trace requirements to test cases, optimize test suites and generate test cases based on rules. Thus, ADPART addresses the testing needs of the business users such as SMEs as well as testers and other technical users.

The first step in using ADPART is to draw the Activity Diagram which accurately reflects the functional flow of the application under test (AUT).

Once the Activity Diagram is drawn, ADPART automatically generates test scenarios, allows the tester to attach criticality to the scenarios and generates test cases.

Consider a shopping website, where a user has various options to add an item to cart. The requirement document and the activity diagram drafted after analyzing the requirements are attached above. The flow of each option is drafted in the diagram.

From this chapter you would have learnt

Overview of Requirements

Details of ACTIVITY diagrams.

Drafting an activity diagram

Deriving test cases using an ACTIVITY diagram

CASE STUDY – for detailed explanation