**UAT Planing**

# 

# Role of the BA in User Acceptance Testing

What is User Acceptance Testing?

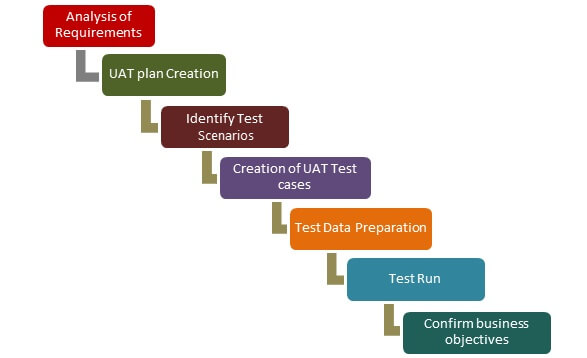
UAT (User Acceptance Testing) is unique among the activities we undertake on our projects, because its focus is on people who are outside the team, and its primary purpose is not finding defects.

**User** – The people who will actually use the solution are not generally active in the project that creates it.  This is true even in Agile projects, where the Product Owner (who is quite involved) is only one person, and the other users remain more distant.  UAT gives users the opportunity to see what has been built and provide feedback to the development team before it is deployed.

**Acceptance** – No system is perfect, so we must ensure that our system is acceptable before it is deployed.  It is precisely the Customer for whom the system is being built and the ultimate end users (those people who are not very involved in the project most of the time) who must accept what we deploy.  UAT is their opportunity to determine if they can indeed accept what has been built.

**Testing** – Written requirements, drawings, mock-ups and the like are not the system, they are merely attempts to clarify what the system will be and do ahead of time.  When it comes to acceptance, those things, and even demonstrations of the system there are no substitute for the Customer and the Users actually putting their hands on the system and trying it out (testing it).

User Acceptance Testing is when the people who will ultimately use the system to do real work run it through its paces in a test environment to be sure that it meets the business need well enough to be accepted.



# Planning UAT Tests

Test Planning for UAT is different from planning the other types of testing on our projects because its purpose is different.  All of the other testing we do is focused on finding defects, so by the time we get to UAT there should be few defects remaining to be found.  The focus of UAT should be on how the system will actually be used in practice.

There are two quite common shortcuts to UAT planning that we must avoid:

1)     The first is to simply reuse the tests that our QA Testers wrote.  That is wasteful of the UAT testers’ time because those tests have already been run and passed.  It is also inappropriate because those tests are focused on finding defects (the purpose of system testing), not judging acceptability (UAT’s purpose).

2)     The other mistake is to not plan UAT tests at all – to just tell the UAT Testers to try the system out.  Since UAT testers are rarely testers by training, this sets them up for not achieving the purpose of UAT.

Because the Business Analyst learned about the business needs and business activities during Requirements Elicitation and Analysis, he/she has an understanding of what should be tested during UAT and should be involved in planning the UAT tests.  UAT planning should go something like this:

1. Create a separate test plan for each type of User who will participate in UAT (both the Active and the Passive users).  The test plan for each type of User must include each business scenario that role will use the system to work through.

Be sure to include not just the normal “happy path” scenarios, but also all of the special cases and error cases. (Error cases would include the user making a mistake, the user encountering mistakes other people have made, and things that can go wrong with the system or other systems.

2. Give the tester appropriate instructions for testing each business scenario.  “Appropriate” instructions ensure that the tester tests all of the things he or she should test, but they don’t include so much detail that the tester could mindlessly follow them without truly evaluating acceptability of the system.

A note on Planning UAT Tests :--

For the new functionality of the system, the instructions should include step-by-step how-to, and should prompt the tester to evaluate if that way of doing the function is good from a business usage perspective.

* For existing functionality, the instructions should just tell the tester what to do (prompting the tester to use his or her existing knowledge about the system and the business process).
* In any test where the data is important to the test (e.g. for special or error scenarios), the instructions should be precise about the data that the tester must use.  In other cases, choice of data should be left to the tester.
* In all cases, the tester should be prompted to check these things:
  + That the system operates as required and handles data and computations correctly. (Defects may have escaped detection in prior testing.)
  + That the way the system operates and presents information will work with how the user’s business processes flow.
  + That the usability, performance, security and other non-functional attributes are appropriate to the business scenario.

**SDLC based UAT Plan**

In **Agile** methods (like Scrum), the software is developed in small Sprints of only a few weeks.  This provides the option of doing mini-UATs (involving appropriate users) after certain Sprints to evaluate chosen functionality.

* Traditional (non-Agile) projects are often broken into **Phases** of a few months each, with certain functionality being built in each phase.  If this approach is being used, each project Phase should end with UAT.
* Even when the project is not broken into Phases or Sprints, a **Function-at-a-Time** development approach would allow UAT of each function as it is completed.
* A variation of the Function-at-a-Time development approach is the **Walking Skeleton** approach in which the skeleton of the software is built first.  (The skeleton would include the product’s structure, menus, and place-holders for each window, screen & dialog.) After UAT of the skeleton, the development team adds a function-at-a-time to the skeleton, allowing Function-at-a-Time UAT.
* Even if none of these development approaches is being used, it is still valuable for users to be given the opportunity to review screens, menus, computations, reports and other things as they are developed.

Resources :-- Alan Koch