QA Testing Manual

The QA Team will concentrate efforts around testing OpenMRS 2.x releases and testing features developed in sprints. Release testing will be announced prior to any OpenMRS 2.x release on https://talk.openmrs.org/c/developers/qa. It will typically begin as soon as a pre-release version i.e. alpha or beta is ready. Testing during development sprints will happen between releases. Please see the QA Team's Wiki page or see the QA Team Trello Board to stay informed on any ongoing QA work!

We use Zephyr for JIRA to track the testing progress. To get familiar with the tool please see this webinar.

In short you can find there test cycles, which group individual test cases. Test cycles are created for each release and sprint. In addition we have the Automated test cycle, which groups tests executed by our CI and the Ad Hoc test cycle, which groups tests not assigned to any other test cycle. Test cycles are created by project leaders, who specify the testing environment and the version of the software that will be tested (see the rightmost columns). You can expand test cycles to see individual test cases by clicking on > in the leftmost column.

QA Processes

Below we describe QA processes.

1. The QA work is organized in test cycles, which are executed in test sprints. Test Cycles should follow the following rules:
   a. All tests should be grouped in Test Cycles. Your module (project) should contain at least the following Test Cycles: Automated Tests, (project+release number) Release Tests and (project+release number) Regression Tests
   b. Add a Test Case to Test Cycle like described in How to add a Test Case to Test Cycle
   c. All automated tests should be added to Automated Tests Test Cycle
   d. Release Tests Test Cycle should contain functional Test Cases that should all have 'Pass' status before next version can be released
   e. Each version of OpenMRS should have separate Release Tests Test Cycle
   f. Regression Tests Test Cycle should contain test cases regarding issues that have already been fixed in order to check if they didn't reappear
   g. Each version of OpenMRS should also have separate Regression Tests Test Cycle
   h. If you need to add a new Test Cycle, follow instructions in How to create a Test Cycle

2. Sprint Tests are test cycles created for specific development sprints. Sprint Tests should follow the following rules:
   a. Before finishing each sprint a Sprint Tests Test Cycle needs to be created concerning actual sprint
   b. Name of the Test Cycle should include both the Sprint name/number and "Sprint Tests" char sequence
   c. A Sprint Tests Test Cycle should contain Test Cases covering all the functionalities developed during the sprint
   d. If you already have a Sprint/Release Tests Test Cycle and wish to create another one concerning a different sprint, Clone already existing Test Cycle instead of creating a new one.
   e. In the cloned Test Cycle you automatically have all the Test Cases from the original
   f. Next, you need to verify whether they are all up-to-date concerning software changes, and if not, simply remove the invalid Test Case from the cloned Test Cycle
   g. Also, some new Test Cases should be added if the sprint you wish to cover contains new functionalities or a major functionality change that requires new Test Cases
   h. If you clone a Sprint Tests Test Cycle from Release Tests Test Cycle, you need to remove Test Cases concerning upgrading to released version, installing released version
      and other release-specific Test Cases

3. Release Tests are test cycles created for specific releases. Release Tests should follow the following rules:
   a. Before each release, a Release Tests Test Cycle needs to be created concerning this concrete release
   b. Name of the Test Cycle should include both the Release name/number and "Release Tests" char sequence e.g.: "OpenMRS 2.2 Release Tests"
   c. A Release Tests Test Cycle should contain all the Test Cases from Sprint Tests Test Cases concerning sprints included in release, upgrading to released version, installing released version, and release-specific Test Cases described in Release Road Map/Manual not covered in sprints
   d. If you already have a Release Test Cycle and wish to create another one concerning a different release, Clone already existing Test Cycle instead of creating a new one.
   e. In the cloned Test Cycle you automatically have all the Test Cases from the original. Next, you need to verify whether they are all up-to-date concerning software changes, and if not, simply remove the invalid Test Case from the cloned Test Cycle
   f. Also, some new Test Cases should be added if the release you wish to cover contains new functionalities or a major functionality change that requires new Test Cases

4. Regression Tests are test cycles created for specific releases, which cover bug fixes. Regression Tests should follow the following rules:
   a. Regression Tests Test Cycle should include Test Cases concerning concrete bug issues that had already been fixed before a release
   b. Name of the Test Cycle should include both the Release name/number and "Regression Tests" char sequence e.g.: "OpenMRS 2.2 Regression Tests"
   c. A Regression Tests Test Cycle concerning a concrete release should include Test Cases concerning issues fixed in that release or in previous releases.

OpenMRS Sync 2.0 Testing

I. Creating a Test Case in OpenMRS Sync

In order to create a test case you need to have an OpenMRS ID. If you do not have one please follow this link.
1. Go to the testing tab at [https://issues.openmrs.org/browse/SYNCT-109?jql=project%3D%27SYNCT%27%20AND%20%20AND%20labels%3D%22sync2%22](https://issues.openmrs.org/browse/SYNCT-109?jql=project%3D%27SYNCT%27%20AND%20%20AND%20labels%3D%22sync2%22)
   From the Menu choose 'Test' > 'Create a test'.

- Describe a Test Case in the 'Summary' field. Summary must include functionality name and short summary of the Test Case (in case of complicated Test Cases)
- In the 'Description' field you can also add more details e.g. user story or leave it empty if 'Summary' field provides enough information
- In 'Attachment' field you can add screen shot or short movie to help localize the issue
- 'Label' field should contain key words used for search purposes
- If the Test Case is meant for an automated test, add a link to automated test code in 'Automated' field

2. Click 'Create' to create the Test Case. Once a Test Case is created, add steps on how to perform it in 'Test Details'
II. Executing a Test Case in OpenMRS Sync

In order to execute a test case you need to have an OpenMRS ID. If you do not have one please follow this link.

2. For any Test Case that doesn't have any executions click on 'Execute' button.
4. Perform all steps shown in the 'Test Details' list on the environment you have opened up in the 3rd step.

If one of the steps in Test Case fails then change its status to "FAIL" both in the Test Execution 'Execution Status' field and in 'Status' column of the appropriate step. Create an issue(or add an existing one if you know it already exists) in the Sync 2 project describing the failure and add it in the 'Defects' column. If you are unsure if the step fail is actually a bug, write in a new topic about it on OpenMRS Talk to the 'developers' list. Describe the failure in both 'Comment' column and Test Execution field. If Test Case passes change Test Execution 'Execution Status' field status to "PASS". 'Execution Status' "PASS" means that all test steps have passed (there is no need to set the "Pass" status on each step in such a case).

5. If you have changed the Test Execution Status to "FAIL", click the 'Return to Test' link on the top right of the page in order to move to the Test Case. Choose "Link" from 'More Actions' menu to link the Test Case with issue added in 'Defects'.

Choose "is a test for" in 'This issue' field and click the 'Link' Button. You also need to change the linked issue status to "Ready for development" if there was no attempt of fixing it or "Test successfull/Test unsuccessfull" if it's supposedly fixed and waiting for test.

III. Priorities in testing

- focus on improvements regarding Sync Module
- handling potentially problematic scenarios
- testing resolving conflicts
- when an error occurs, test how application handles it and comes back to normal state

-- TODO ---