**How do you decide when you have 'tested enough'?**

1. Deadlines (release deadlines, testing deadlines, etc.)
2. Test cases completed with certain percentage passed.
3. Test budget depleted.
4. Coverage of code/functionality/requirements reaches a specified point.
5. Bug rate falls below a certain level.
6. Beta or alpha testing period ends.

Deadlines (release deadlines, testing deadlines, etc.

Testing done is enough when :

1. 100% requirements coverage is achieved.  
2. More than 95% of test coverage and 100% functional coverage is achieved.  
3. When we achieved the target time.  
4. All showstopper and Major defect are identified, verified and closed.  
5. Less than 5%Minor defect are open, and if open work around is available.  
6. All defects are retested and closed.  
7. All corresponding regression scenario of retested and closed defect are also tested.  
8. All critical testcases are passed  
9. All test document and deliverables are prepared, reviewed and published across.  
10. Sign off is given

As of my knowledge following parameters should be considered to decide an S/W tested enough

1) Bug Rate

2) Schedule

3) Budget

4) Test Case Coverage

5) Risk Analysis ( Means all risk areas covered or not)

Anyone criteria of above will not serve the purpose. Based on company priorities we have to take optimal decision by considers all the above parameters.

How do you decide when you have tested enough?

How to know when stop testing

It's a never-ending process. Still, the testing process can be stopped

(1) Based on the Testing deadline and Release deadlines

(2) Based on the number of test cases covered with a certain percentage passed

(3) Based on the coverage of code, functionality,module, requirement reaches to the specified level

(4) When the bug rate falls to down

(5) When the alpha or beta testing period ends...

As of my knowledge following parameters should be consider to decide a S/W tested enough

1) Bug Rate

2) Schedule

3) Budget

4) Test Case Covarage

5) Risk Analysis ( Means all risk areas covered or not)

Any one criteria of above will not serve the purpose.Based on company priorities we have to take optimal decision by considers all above parameters

...............................................................

we can decide same on following bases.

1) We define Pass Criteria in Test Plan and if Pass Criteria matches with our scedule or current scenario we are in situation to say that we have tested enough.

2) I t can be decided by , how many priorities items are left to test, how many are already tested, and how much time will be taken for doing the same,

we match all these with our Pass Criteria then discuss withour concerned manager and can take decisson that we have tested enough.

............................

You have tested enough when the company's Exit Criteria have been met. This typically includes (not an exhaustive list):

1) All test cases have been run or are otherwise accounted for

2) There are no blocked or partially completed test cases.

3) All test cases have passed unless they are reported as defects

4) There are no 'Open' defects

5) All showstopping defects must be 'Closed'

6) The number of 'Unresolved' defects by Severity and Priority meet the thresholds which were agreed upon by all relevant stakeholders in the client and provider organizations

………………………………………….

Stop the Testing when the test cases have been completed with some prescribed pass percentage.

 Stop the Testing when the testing budget comes to its end.

Stop the Testing when the code coverage and functionality requirements come to a desired level.

Stop the Testing when bug rate drops below a prescribed level

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

Preparation of predefined number of test cases ready before test execution cycle.

b) Execution of all test cases In every testing cycle.

c) Stopping the testing process when all the test cases get Passed

d) Alternatively testing can be stopped when percentage of failure in the last testing cycle is observed to be extremely low.

\\\\\\\\\\\\\\\\\\///////////////////////////////////\\\\\\\\\\\\\\\////////////\\\\\\\\\\\\\

Coverage of the code is good  
2) Mean time between failure is quite large  
3) Defect density is very low  
4) Number of high severity Open Bugs is very low.

Here ‘Good’, ‘Large’, ‘Low’ and ‘High’ are subjective terms and depend on the type of product being tested. Ultimately, the risk associated with moving the application into production, as well as the risk of not moving forward, must be taken into consideration.

**Question2**

Testing is complete when you have stopped finding bugs. Or, more practically, when the rate at which you find new bugs greatly slows down.

Bugs tend to "cluster" in certain modules and certain functions: The moment you find a bug in one, you know that you should look in it further for more bugs. To find bugs, you can [use the techniques](https://softwareengineering.stackexchange.com/questions/192/is-test-coverage-an-adequate-measure-of-code-quality/14660#14660) of blackbox testing, whitebox testing, and mutation testing. **As long as you are finding bugs, you know that your testing process is working!**

To visualize your progress, chart the number of bugs your team has found per day. If the chart slopes down, then you know the techniques your team is using aren't going to find them anyway. Of course, if you believe your techniques aren't up to par, then please read Myers's book an apply the principles.

Now, there is a chance that you might just be short of missing a new patch of bugs, and the rate of finding bugs would have greatly increased had you continued testing for a little bit more. However, if you believe your techniques are sound, this is unlikely.

**………………………………………………………………………………………………….**

I think this is referred to as striping but don't quote me on that.)

And then the final thing to think about is not what you test but what you actually fix when issues are discovered. It's common to say "fix all bugs" but the reality is that there are time pressures and not all bugs are equal. Again, regular bug scrubs with all the relevant parties are the best way forward. This is particularly relevant where a bug fix may be particularly intrusive as the additional work in retesting and regression testing it generates may outweigh the benefit of the fix.

**………………………………………………….**

**Example:**

One useful technique to think about is that you may also vary the use cases that are tested with each release. For instance you might have a list of non-critical test cases and test half of them with one release and half with the next (then alternate). This way you're increasing the total test coverage you get for the effort (though at the risk of regression bugs being introduced).

This could also extend to platform testing - if you support two database back ends (or multiple browsers) test half the app on one, the other half on the other and then swap next release.

**Question2**

**7 Things that are important in Bug report or reporting a Bug.**

***-****Title  
- Steps to reproduce  
- Expected result  
- Actual result  
- Attachments  
- Impact Area  
- Priority / Severity*

So a Proper Bug report should have these essential points.  
Let understand these points with the example.

**Title** —  
Just be Be Specific what you write(t should say all about what comes in the bug), Make it little descriptive if possible.

Bad Example : Login button error.  
Good Example : Post call validation error on login page with valid user.

**Steps to reproduce** —  
Be concise and give a proper step by step way to reach to the bug state.

1. Navigate to the Login page.
2. Click on the username field and enter a valid username.
3. Click on the password field and enter valid password.
4. Click on the Submit button.

Similarly **Expected result , Actual results** are another import points that should not be missed while adding a bug to any test management tool.

**Impact Area —**This is most of the time missed by QA or Testers but be sure to add this in your bug report every time.

**Priority —**You have to set the Priority , bug can be a

* Blocker
* Critical
* Major,
* Minor, Low, or a suggestion.

A Bug priority from P0 to P5 can be given so that the important ones are viewed first.

**Severity —**

Types of Severity:

* Blocker: Testing blocked due to this issue.
* Critical: Loss of data and Application not working
* Major: Major loss of function.
* Minor: Minor loss of function.
* Low: Some UI enhancements or Non blocker.
* Enhancement: Some enhancement in the existing one.

Other important points to include in the Bug report are

* **Screenshot**
* **Environment**working on ( Test app or Production.
* **Platform & Version (Browser extra).**

**………………………………………………………………………………………….**

* 1. SUMMARY
  2. A summary is a short description of an error. Ideally, it should answer three questions: what, where, and when. For example: (what?) Console Error appears (where?) on the Statistics tab (when?) after a user clicks the Download button. Sometimes, however, you can skip the where part. For example: an application crashes after a user clicks the Sign In button. You can specify the page where it happens or skip this part if there is only one place to find the form and the mention of it is self-explaining.
  3. PRODUCT This part usually indicates a version of a build under test that features a bug you are describing. A QA specialist provides this information so that a developer can compare the latest tested version of a product with a previous one and see what has changed. It makes it much easier to identify what caused the defect.
* 3. PLATFORM
* A software tester should mention on what platform the defect appears. For desktop projects, we indicate an operating system. For web projects, it is browser information that matters the most. As for mobile projects, a QA specialist should indicate a device model and a version of an operating system it uses.

1. STATUS

A bug status keeps all members of the development team updated on the bug fixing process. The status lets you know whether a developer has accepted the bug, whether it has been sent for retesting, fixed and closed, etc. The variety of statuses depends on the workflow a team uses.

1. PRIORITY

Bug priority shows how critical the defect is for the business and determines the order of fixing. Usually, a Product Manager, a Product Owner, or a Team Lead assigns the priority. There are three levels of priority: P1 – High: needs to be fixed immediately. P2 – Medium: needs to be fixed after high-priority defects. P3 – Low: fixed in the last instant, after there are no P1 and P2 defects left.

1. SEVERITY

Unlike the priority, the severity is assigned by a software tester based on how dangerous a bug is for the system, to what extent it affects the critical functionality, etc. S1 – Blockers: it is impossible to use an app due to the defect. S2 – Critical: a defect blocks a part of the functionality, but there is an alternative way to use it. S3 – Medium: a defect affects a part of the functionality, so that a feature works, but not correctly. S4 – Minor: a defect doesn’t interfere with the core functionality, but rather with UI/UI aspects. S5 – Trivial: a defect has a minimal impact on the overall functionality or performance of a software product, like a misspelled word or a grammatical error.

PRECONDITIONS

Preconditions describe the necessary actions or parameters to execute or meet before executing the steps that let you recreate a defect. No particular format is required for this description, just make sure to keep the list and order logical.

STEPS TO REPRODUCE

The best way to describe the steps is to provide a numbered list with the sequence of user actions that end with discovering a defect. Use simple sentences to suggest what to do next. For example: A user opens the Statistics tab. The user clicks on the Save button. The user refreshes the page. … and so on.

ACTUAL RESULT

An actual result is a problem that happens after a person follows the steps mentioned above. Describe the result using the rule mentioned in the summary, stating what happened, where, and when. It will help a developer understand what the problem is. Besides, such concise and accurate descriptions will also be useful for a QA team in the future.

EXPECTED RESULT

In this paragraph, describe an expected outcome of the listed steps – in other words, how the application is expected to behave. That’s why an expected result can imply an error – in case a QA engineer checks a negative scenario. For example, if a user enters incorrect credentials, they cannot sign in to the system and see an error message.

11. ATTACHMENTS

Attachments are extras that can come in a bug report. It is often easier to reproduce a defect using visual guidelines, especially if it is complicated to describe the steps or the result verbally. Screenshots or short videos attached to the textual description help to prevent any misunderstanding. Just remember that visual materials should be relevant and clear.

**Bug life cycle includes following steps or status:**

1. **New:** When a bug is reported and posted for the first time. Its state is given as new.
2. **Assigned:** After the tester has reported the bug, the lead of the tester confirm that the defect is valid and it assigned to the appropriate developer or developers team.
3. **Open:** It means that the developer has begun to analyze the bug and trying to fix.
4. **Fixed:** After developer changed the code and fixed bug, they change state to “Fixed” and it can be pass to QA team for retesting.
5. **Pending Retest:** At this stage bug report waiting for retesting.
6. **Retest:** At this stage, the testers check the amendments and retest the changing that developers have made.
7. **Verified:** If retesting it isn’t detected the bug and the product is working properly, tester changes the bug report status in “Verified”.
8. **Reopen:**Reopen: In case the tester rechecked and the bug still exists, the state of bug becoming “Reopen” and bug report goes through the life cycle once again.
9. **Closed:** Once the developer has corrected the mistakes, he sends the product to testers for retesting. If the tester decides that the bug is fixed, he or she changes the bug report status to “Closed.” This means that the defect is fixed, checked and approved.
10. **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “duplicate**“.**
11. **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “rejected”.
12. **Deferred:** this means that the bug will be fixed, but in another release, and now it’s waiting. Usually, the reason for this is the low priority of bugs and lack of time.
13. **Not a bug:** Bag report can have that status, in the case of, for example, if a customer asked to make any little changes to the product: change colour, font, and more.

**Bug id:it is a unique number given to the bug.**

**Test case name:**

when we find a bug,we send a bug report, not the test case to the concerned developer.it is used as a reference for the test engineer.

**Severity:**

**It is the impat of a bug on the application.It can be a blocker, critical, major, and minor.**

**priority:**

**In this we have to decide which bug has to be fixed first.It could be P1/p2/p3/p4,**

**urgent,high, medium, and low.**

**status:**

**the different status of the bug which could be assigned,**

**Invalid,Duplicate,Deferred, and so on.**

**Reporter:**

In this, we will mention the name of the person who found the bug .It could be the test

engineer, and sometime it may be a developer, business analyst, customer, etc

**Date:**

**It provides the date when the bug is found.**

**Release/Build version:**

**It provides the release number in which the bug occurs, and also the build version**

**of the application.**

**platform:**

**mention the platform details,Where we exactly find the bug.**

**Description:**

**In this,we will explain the navigation steps,expected and actual results of**

**the particular bug.**

**Attachments:**

**Attach the screenshots of the bug,which we captured because it helps the developers**

**to see the bug.**

**severity:**

**The impact of the bug on the application is known as severity.**

**It can be a blocker,Critical,major and minor for the bug.**

**Blocker:**

**If the severity of a bug is blocker,Which means We cannot proceed to the next module,**

**and Unnecessarily test engineer sits ideal.**

**There are two types of blocker bug,which are as follows:**

**A major feature is not working:**

**Login To HDFC,Amount transfer is not working**

**The major flow is not working:Login and sign up itself Not working in HDFC application.**

**Severity:**

**Critical:**

**if it is critical,THat means the main functionality is not working,and the test engineer**

**cannot continue testing.**

**Major:if it is major,which means that the supporting components and modules are not working fine,but test engineer**

**can continue the testing.**

**Minor:**

**if the severity of a bug is major,which means that all the U.I Problems**

**are not working fine,But testing can be processed without interruption.**

**priority:**

**Priority is important for fixing the bug or which bug to be fixed**

**first or how soon the bug should be fixed.**

**It can be urgent,high, medium, and low.**

**High:**

**It is a major impact on the customer application, and it has to be fixed first.**

**Medium:**

**In this, the problem should be fixed before the release of the current version in development.**

**Low:**

**The flow should be fixed if there is time,but it can be deferred with the**

**next release.**

**///////////////////////////////**

**…………………………………………………………………………………………………………………**

#### **2. Environment**

A bug can appear in a particular environment and not others. For example, a bug appears when running the website on Firefox, or an app malfunctions only when running on an iPhone X. These bugs can only be identified with [cross browser testing](https://www.browserstack.com/live) or cross device tests.

When reporting the bug, QAs must specify if the bug is observed in one or more specific environments. Use the template below for specificity:

* **Device Type**: Hardware and specific device model
* **OS**: OS name and version
* **Tester**: Name of the tester who identified the bug
* **Software version**: The version of the software which is being tested, and in which the bug has appeared.
* **Connection Strength**: If the bug is dependent on the internet connection (4G, 3G, WiFi, Ethernet) mention its strength at  
  the time of testing.
* **Rate of Reproduction**: The number of times the bug has been reproduced, with the exact steps involved in each reproduction.

#### **. Expected Result**

How is the software supposed to work, with regard to the particular area in which the bug appears? The developer needs to know what the requirement is, in order to gauge the extent to which the bug is disrupting the user experience.

Describe the ideal end-user scenario, and try to offer as much detail as possible. Don’t just leave it at “the app is crashing, and it shouldn’t.”

**Actual Result**

Detail what the bug is actually doing, and how it is a distortion of the expected result.

* Elaborate on the issue
* Is the software crashing?
* Is it simply pausing in action?
* Does an error appear?
* Or is it simply unresponsive?

Specificity in this section will be most helpful to developers. Emphasize distinctly on what is going wrong. Provide additional details so that they can start investigating the issue with all variables in mind. For example:

* “Link does not lead to the expected page. It shows a 404 error.”
* “When clicked, the button does not do anything at all.”
* “The main image on the homepage is distorted on the iPhone X.”

#### **Visual Proof of Bug**

Screenshots, videos of log files must be attached to clearly depict the occurrence of the bug. Depending on the nature of the bug, the developer may need video, text, and images.

Testing using BrowserStack can leverage multiple debugging options – text logs, visual logs (screenshots), video logs, console logs, and network logs. These make it easy for QAs and devs to detect exactly where the error has occurred, study the corresponding code and implement fixes.

**Bug Severity**

Every bug must be assigned a level of severity and corresponding priority. This reveals the extent to which the bug affects the system, and in turn, how quickly it needs to be fixed.

Levels of Bug Severity:

* Low: Bug won’t result in any noticeable breakdown of the system
* Minor: Results in some unexpected or undesired behavior, but not enough to disrupt system function
* Major: Bug capable of collapsing large parts of the system
* Critical: Bug capable of triggering complete system shutdown

Levels of Bug Priority:

* Low: Bug can be fixed at a later date. Other, more serious bugs take priority
* Medium: Bug can be fixed in the normal course of development and testing.
* High: Bug must be resolved at the earliest as it affects the system adversely and renders it unusable until it is resolved.

### Date and Time

The date and time that the defect occurred or reported is also essential. This is normally useful when you want to search for defects that were identified for a particular release of software or from when the testing phase started.

### Related requirement

Essentially, all features of a software application can be traced to respective requirements. Hence, when a failure is observed, we can see what requirements have been impacted.

This can help in reducing duplicate defect reports in that if we can identify the source requirement, then if another defect is logged with the same requirement number, we may not need report it again, if the defects are of similar nature.

* Unique report number (ID)

Each bug report must have a unique identifier. It usually consists of the first letters of the project’s title and a serial number. Most bug tracking systems automatically assign an ID for each bug.

* Author

Most bug tracking systems have this information by default, so you could quickly contact the tester who found and described the bug.

* Assigned to

This section specifies the information about the developer who will fix the bug. Depending on the project's agreements and organization, the bug can be assigned to a senior QA or PM who will decide which developer should fix it.

* The status that shows the current stage of working with the bug