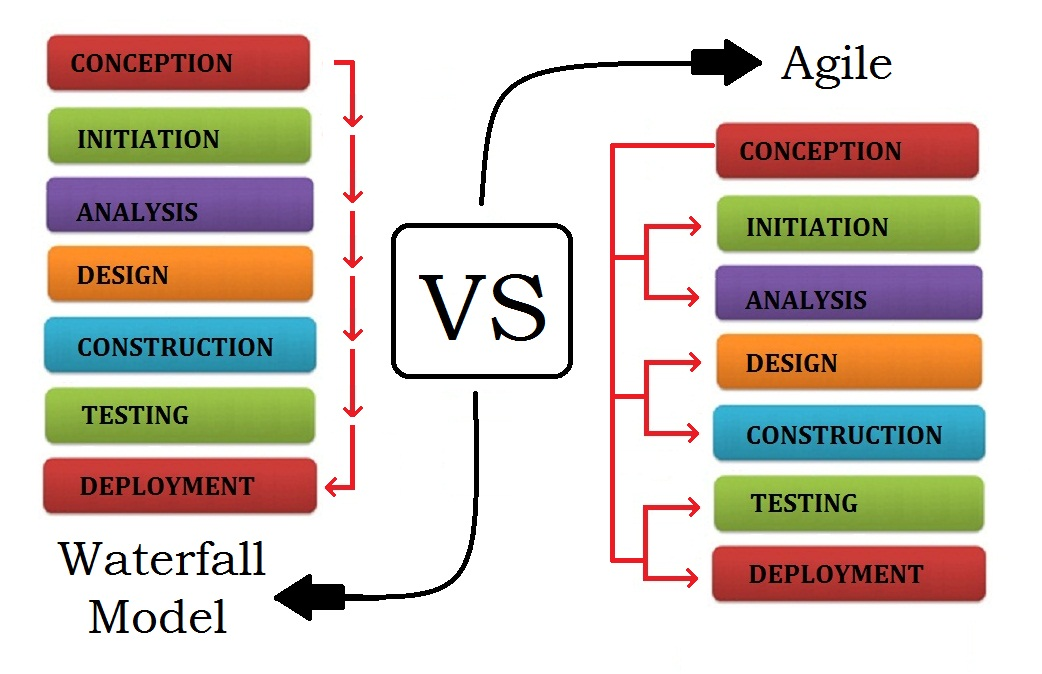
**SDLCSTLC**



Test Plan

* Scope(In & Out) of this testing
* Approach of testing
* Schedules & Tasks
* Test strategy
* Entry Criteria
* System Requirements
* Exit Criteria
* Risks involved
* Reviews & Approvals

**Entry Criteria**:

* Requirements are Freezed and documented
* Design and development is completed
* Unit Testing is Signed off
* Test Proposal is approved
* Test Plan is reviewed & signed off by PM & BA
* Run Plan & Test data are set up
* System & Network requirements are satisfied
* Required number of test resources are in place
* Test Environment is stable
* Smoke/Sanity test is PASS

**Functional team sign off happens when**

* **Exit Criteria is satisfied**:
* All Show Stopper(Severity 1) & Critical(Severity 2) defects are Resolved (FIXED and Deployed)
* All Test cases are executed >> 100% execution completed
* No Test cases are blocked or on hold due to some defects
* Cosmetic/Low priority defects are reviewed with BA/PM and good to go >> Can we live with it >> if YES then go ahead
* Test phase is signed off and approved

**manual and automation test plan**

****

**Automation Testing Test Plan Sections**

Section #1: Scope -small , huge regression

Choose the frequent test cases/scenarios that are to be regressed over and over across multiple cycles.

Section #2: Test strategy / Framework

This section is referred to as Framework in the automation world. Some frameworks are extremely challenging to create and also are effective – but time, effort and competency wise they are demanding**.** Always look for a middle ground and do the best you can without jeopardizing over utilization of resources.

Decide on coding best practices to be used, naming conventions, locations for test assets to be stored, format of test results , etc. to maintain uniformity and increase productivity.

Section #3: Resources/roles and responsibilities-Team, automation

The first step in this direction is to understand the team’s capabilities and anticipate ahead the scope of automation coming into picture. This will help choose a team that suits both the automation and manual testing needs.

Choose a team well versed with AUT, test management, defect management and other SDLC activities

Section #1: Scope

Section #4: Tools

Pick automation tools based on the following rules:

Does the company already have licenses for a certain tool.

Look for open source(but reliable) tools

Section #5: Schedules-short, long

Include time for code-walkthroughs and inspection of the automation scripts

Maintain the scripts on a timely basis not going to use for the next 6 months later**.**

Section #6: Environment- target AUT and tool compatibility

The target environment that your AUT is going to run and the automation tool that you want to use should be compatible. This is one of the factors to be considered pre-licensing for the tool.

Also, analyze if the rest of the management tools in place and the automation tool you are trying to bring in are inter-connectible for additional benefit.

Section #7: Deliverables- release

Your test scripts are your deliverables. However, not everyone is automation/programming language savvy.

Section #8: Risks

if you propose automating a system, pick the one that is

Stable and not too much maintenance

Has a scope for huge regression suites

Does not have too much of manual intervention

Section #9: Test data- external sources

Take into consideration the security aspects of the data

Do not hard code any test data into the scripts. This just leads to too much script maintenance and might induce errors during modification.

Section #10: Reports/results-technical

Script execution results are also technical and might not be easily understood by the rest of the teams. Plan on writing detailed results to notepad or excel sheets as an additional measure.

Detailed framework documents, review results, defect reports, execution status reports are also expected.

**project documentation(Test script design)**

The goal is to communicate and document the essence of the project, primarily for informational purposes. Typically described by a sentence or two of text in each of the sections of the project.

Plans for scope management should be provided and include procedures for change control.

The Testing Plan should be consistent with the complexity of the project and the associated risks.

The project plan should include a Scope Plan, which includes the project objectives and the project deliverables, and which includes at least a simple Work Breakdown Structure.

Documentation should provide all of the information required to initiate, plan, execute, monitor, and complete the project in a timely and cost-effective manner.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

According to ANSI/IEEE 1059 standard, Testing can be defined as *A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item*.

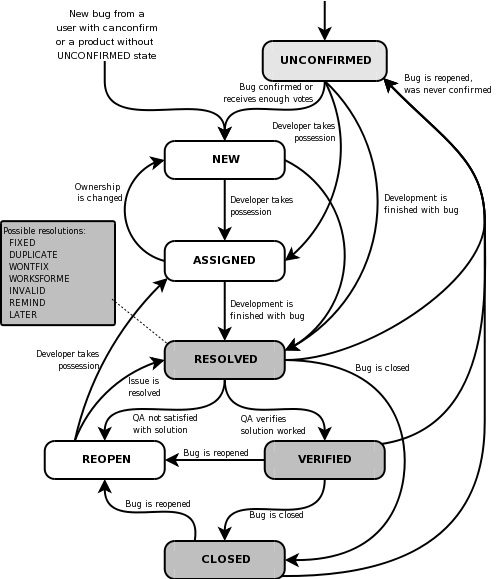
All GUI items, connections with databases, field validations etc can be efficiently tested by automating the manual process.

Bug Tracking

Bug tracking is a process used by quality assurance personnel and programmers to keep track of software problems and resolutions. A bug tracking system is normally put in place to store information about reported bugs. This type of issue-tracking system provides a clear, centralized overview of development requests and their corresponding states.

Bug tracking enables users to enter bug reports directly into a system that logs and tracks them. Diligent use of a bug tracking system provides a record of a software team's effectiveness. Local bug trackers are often used by teams of application support professionals to keep track of issues communicated to software developers.

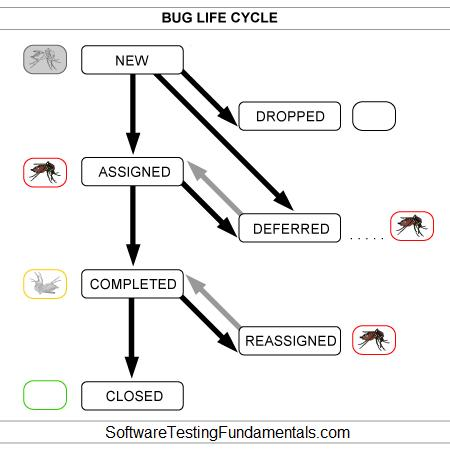
Bug tracking systems include a database, which keeps track of facts pertaining to each bug. These facts might include the time a bug was reported, its severity, incorrect program behavior, details on how to recreate the bug, who reported the bug and what the programmers did to fix it. Bug tracking systems are associated with a bug's life cycle, which is tracked through the status assigned to each bug. This allows administrators to set permissions based on a bug's status, move bugs to other statuses or delete them.



**defect life cycle**

Defect Life Cycle (Bug Life cycle) is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used.

Nevertheless, the life cycle in general resembles the following:



Status Alternative Status

NEW

DROPPED REJECTED

ASSIGNED OPEN

DEFERRED

COMPLETED FIXED, RESOLVED, TEST

REASSIGNED REOPENED

CLOSED VERIFIED

Defect Status Explanation

NEW: Tester finds a defect and posts it with the status NEW. This defect is yet to be studied/approved. The fate of a NEW defect is one of ASSIGNED, DROPPED and DEFERRED.

ASSIGNED / OPEN: Test / Development / Project lead studies the NEW defect and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer’s responsibility is now to fix the defect and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a defect can be open yet unassigned.

DEFERRED: If a valid NEW or ASSIGNED defect is decided to be fixed in upcoming releases instead of the current release, it is DEFERRED. This defect is ASSIGNED when the time comes.

DROPPED / REJECTED: Test / Development/ Project lead studies the NEW defect and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.

COMPLETED / FIXED / RESOLVED / TEST: Developer ‘fixes’ the defect that is ASSIGNED to him or her. Now, the ‘fixed’ defect needs to be verified by the Test Team and the Development Team ‘assigns’ the defect back to the Test Team. A COMPLETED defect is either CLOSED, if fine, or REASSIGNED, if still not fine.

If a Developer cannot fix a defect, some organizations may offer the following statuses:

Won’t Fix / Can’t Fix: The Developer will not or cannot fix the defect due to some reason.

Can’t Reproduce: The Developer is unable to reproduce the defect.

Need More Information: The Developer needs more information on the defect from the Tester.

REASSIGNED / REOPENED: If the Tester finds that the ‘fixed’ defect is in fact not fixed or only partially fixed, it is reassigned to the Developer who ‘fixed’ it. A REASSIGNED defect needs to be COMPLETED again.

CLOSED / VERIFIED: If the Tester / Test Lead finds that the defect is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED. This is the happy ending.

Defect Life Cycle Implementation Guidelines

Make sure the entire team understands what each defect status exactly means. Also, make sure the defect life cycle is documented.

Ensure that each individual clearly understands his/her responsibility as regards each defect.

Ensure that enough detail is entered in each status change. For example, do not simply DROP a defect but provide a reason for doing so.

If a defect tracking tool is being used, avoid entertaining any ‘defect related requests’ without an appropriate change in the status of the defect in the tool. Do not let anybody take shortcuts. Or else, you will never be able to get up-to-date defect metrics for analysis.

<http://softwaretestingfundamentals.com/defect-life-cycle/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<http://www.bugzilla.org/docs/2.18/html/lifecycle.html>

**End-to-End Test**

End-to-end testing is a methodology used to test whether the flow of an application is performing as designed from start to finish. The purpose of carrying out end-to-end tests is to identify system dependencies and to ensure that the right information is passed between various system components and systems.

End-to-end testing involves ensuring that that integrated components of an application function as expected. The entire application is tested in a real-world scenario such as communicating with the database, network, hardware and other [applications](http://www.techopedia.com/definition/7035/end-to-end-test).   
  
For example, a simplified end-to-end testing of an email [application](http://www.techopedia.com/definition/7035/end-to-end-test) might involve:

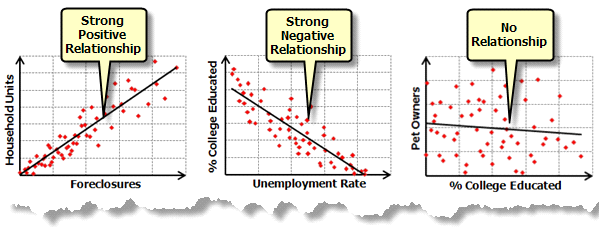
* Logging in to the application
* Accessing the inbox
* Opening and closing the mailbox
* Composing, forwarding or replying to email
* Checking the sent items
* Logging out of the application

[**http://www.techopedia.com/definition/7035/end-to-end-test**](http://www.techopedia.com/definition/7035/end-to-end-test)

**Back-end Testing**

Back-end testing is basically testing things that are on the "back-end" - usually servers and databases. Thus "database testing" can be part of back-end testing. Database testing, in specific, is just what it sounds like: testing a database.

**Functional**, **Negative regression**



<http://www.tutorialspoint.com/software_testing/levels_of_testing.htm>

**4 Types of automated tests - unit, integration, UI, and performance**

A software engineer could spend their life continually improving test automation - it's a big field. While the sky is the limit, there are at least 4 types of automated tests: unit, integration, UI, and performance.

Unit --> Integration: complex integration tests if you have a simple unit test.

Integration --> UI: It's going to be near impossible to do a UI test (which usually has poor APIs) if you can't at least integrate the backend (with at least has APIs - like web services, SQL, or Java, C# calls).

UI --> Performance: If you can't at least functionally run the code from end-to-end, then you can't expect reliable performance measures on it. Yes, there are always exceptions, and semantics (one may consider "UI" to be fronted integration, or one may test performance on just the backend APIs and bypass the UI). But in general, these 4 tests are a very natural path to follow.

The higher you go, the more expensive: Unit tests (low-level) are cheapest, performance tests (high-level) are most expensive. So it's bad business to pay for an integration test to do the work of a unit test.

These 4 types of tests should be separated. You can call any code from a unit test (depending on security, you could even call APIs to shutdown the server), so you could mix all your tests into one test harness. But don't do this - it will burn you. For example, unit tests are generally fast (they're all in-memory), whereas UI and integration are much slower (databases, web services, IIS hosts, etc...) So you don't want them coupled together because the slow integration tests will bog down your fast unit tests, and then developers never run unit tests before check-in because "it takes too long".

Unit testing is one tool. There is different types of code (algorithms, data containers, plumbing, installation scripts, UI, persistence plumbing, etc...). This requires an arsenal of developer skills, of which unit testing is one tool. With respect to unit testing and code coverage, the goal isn't N% coverage of all code, but rather N% coverage of unit-testable code. (You can get better coverage tools, like NCover, which can provide coverage when running integration, UI, and even manual tests run by QA, but that's a different story).

<http://www.timstall.com/2011/02/4-types-of-automated-tests-unit.html>

Load Testing vs Performance Testing

Performance Testing = how fast is the system?

Load Testing = how much volume can the system process?

Performance testing seems to be much broader than load testing.

Load testing is usually focused on metrics like requests per second and concurrent users (the cause); whereas performance testing is more concerned with response times (the effect).

**System integration testing**

System integration testing (SIT) is a high-level software testing process and can operate in coordination with other systems in the same environment – target AUT and tool compatibility. The testing process ensures that all components and subcomponents are integrated successfully to provide expected results. Testers verify that all related systems in the same environment maintain data integrity in the testing process.

SIT validates [data](http://www.techopedia.com/definition/24590/system-integration-testing-sit) integrity between the different sub-components that make up a specified system**.** The SIT process occurs after unit testing and before validation testing. Due to the fact that SIT concentrates on testing the dependencies between sub-components, it is often subjected to regression testing scenarios. Regression testing facilitates the addition of new test cases. From an application perspective, SIT testing focuses on access to actual data passed through the components and initial connectivity.   
  
The main goal of SIT testing is to test the automation of aggregated components and the dependencies that exist between them.

<http://www.techopedia.com/definition/24590/system-integration-testing-sit>

**System integration testing**

System integration testing is testing performed when two systems, generally presumed stable themselves, are integrated with one another. For example, this could be when an inventory management system is integrated with a sales accounting system. Each system feeds into the other. The goal of systems integration testing is to ensure that the data crossing the boundary between systems is received, stored and used appropriately by the receivient system. Until integration begins, testing of the isolated systems is done on mocked or replayed data and not on "live" data. Integration testing is the final step before customer acceptance testing.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.

Regression testing is so important because of the following reasons:

Minimize the gaps in testing when an application with changes made has to be tested.

Testing the new changes to verify that the change made did not affect any other area of the application.

Mitigates Risks when regression testing is performed on the application.

Test coverage is increased without compromising timelines.

Increase speed to market the product.

User Acceptance Testing (UAT)

User acceptance testing (UAT) is the last phase of the software testing process or STLC. During UAT, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications.

UAT is one of the final and critical software project procedures that must occur before newly developed software is rolled out to the market.

UAT is also known as beta testing, application testing or end user testing.

UAT directly involves the intended users of the software. UAT can be implemented by making software available for a free beta trial on the Internet or through an in-house testing team comprised of actual software users.

Following are the steps involved in in-house UAT:

Planning: The UAT strategy is outlined during the planning step.

Designing test cases: Test cases are designed to cover all the functional scenarios of the software in real-world usage. They are designed in a simple language and manner to make the test process easier for the testers.

Selection of testing team: The testing team is comprised of real world end-users.

Executing test cases and documenting: The testing team executes the designed test cases. Sometimes it also executes some relevant random tests. All bugs are logged in a testing document with relevant comments.

Bug fixing: Responding to the bugs found by the testing team, the software development team makes final adjustments to the code to make the software bug-free.

Sign-off: When all bugs have been fixed, the testing team indicates acceptance of the software application. This shows that the application meets user requirements and is ready to be rolled out in the market.

UAT is important because it helps demonstrate that required business functions are operating in a manner suited to real-world circumstances and usage.

<http://www.techopedia.com/definition/3887/user-acceptance-testing-uat>

<http://www.tutorialspoint.com/software_testing/levels_of_testing.htm>

##### Sanity Testing

Sanity Testing is also called **tester acceptance testing**.

**After receiving a software build, with minor changes in code, or functionality, Sanity testing is performed to ascertain that the bugs have been fixed and no further issues are introduced due to these changes.The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing**.

The **objective is "not" to verify thoroughly the new functionality**, but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

## Smoke Testing Vs Sanity Testing - Key Differences

|  |  |
| --- | --- |
| **Smoke Testing** | **Sanity Testing** |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality  / bugs have been fixed |
| The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the  "rationality" of the system in order to proceed with  more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing is usually performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is  unscripted |
| Smoke testing is a subset of Regression testing | Sanity testing is a subset of Acceptance testing |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component  of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |

[**http://www.softwaretestingclass.com/sanity-testing/**](http://www.softwaretestingclass.com/sanity-testing/)

[**http://www.guru99.com/smoke-sanity-testing.html**](http://www.guru99.com/smoke-sanity-testing.html)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

E2E Web Services API

ad-hoc testing

**When a software testing performed without proper planning and documentation, it is said to be Adhoc Testing. Such kind of tests are executed only once unless we uncover the defects.**

**Adhoc Tests are done after formal testing is performed on the application. Adhoc methods are the least formal type of testing as it is NOT a structured approach. Hence, defects found using this method are hard to replicate as there are no test cases aligned for those scenarios.**

**Testing is carried out with the knowledge of the tester about the application and the tester tests randomly without following the specifications/requirements. Hence the success of Adhoc testing depends upon the capability of the tester, who carries out the test. The tester has to find defects without any proper planning and documentation, solely based on tester's intuition.**

**Adhoc testing can be performed when there is limited time to do exhaustive testing and usually performed after the formal test execution. Adhoc testing will be effective only if the tester has in-depth understanding about the System Under Test.**

<http://www.tutorialspoint.com/software_testing_dictionary/adhoc_testing.htm>

GUI testing

GUI testing is a process to test application's user interface and to detect if application is functionally correct. GUI testing involves carrying set of tasks and comparing the result of same with the expected output and ability to repeat same set of tasks multiple times with different data input and same level of accuracy. GUI Testing includes how the application handles keyboard and mouse events, how different GUI components like menubars, toolbars, dialogs, buttons, edit fields, list controls, images etc. reacts to user input and whether or not it performs in the desired manner. Implementing GUI testing for your application early in the software development cycle speeds up development, improves quality and reduces risks towards the end of the cycle. GUI Testing can be performed both manually with a human tester or could be performed automatically with use of a software program.

<http://www.appperfect.com/products/application-testing/app-test-gui-testing.html>

<http://www.softwaretestingsoftware.com/all-types-of-software-testing/#end-to-end> testing

Software Testing Life Cycle

<http://softwaretestingfundamentals.com/software-testing-life-cycle/>

<http://www.guru99.com/software-testing-life-cycle.html>

SOFTWARE DEVELOPMENT LIFE CYCLE

<http://softwaretestingfundamentals.com/software-development-life-cycle/>

Spiral

<http://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Comprehensive Testing Checklist for Testing Web and Desktop Applications:**

**Assumptions:** Assuming that your application supports following functionality  
- Forms with various fields  
- Child windows  
- Application interacts with database  
- Various search filter criteria and display results  
- Image upload  
- Send email functionality  
- Data export functionality

**General Test Scenarios**

1. All mandatory fields should be validated and indicated by asterisk (\*) symbol  
2. Validation error messages should be displayed properly at correct position  
3. All error messages should be displayed in same CSS style (e.g. using red color)  
4. General confirmation messages should be displayed using CSS style other than error messages style (e.g. using green color)  
5. Tool tips text should be meaningful  
6. Dropdown fields should have first entry as blank or text like ‘Select’  
7. Delete functionality for any record on page should ask for confirmation  
8. Select/deselect all records options should be provided if page supports record add/delete/update functionality  
9. Amount values should be displayed with correct currency symbols  
10. Default page sorting should be provided  
11. Reset button functionality should set default values for all fields  
12. All numeric values should be formatted properly  
13. Input fields should be checked for max field value. Input values greater than specified max limit should not be accepted or stored in database  
14. Check all input fields for special characters  
15. Field labels should be standard e.g. field accepting user’s first name should be labeled properly as ‘First Name’  
16. Check page sorting functionality after add/edit/delete operations on any record  
17. Check for timeout functionality. Timeout values should be configurable. Check application behavior after operation timeout  
18. Check cookies used in an application  
19. Check if downloadable files are pointing to correct file paths  
20. All resource keys should be configurable in config files or database instead of hard coding  
21. Standard conventions should be followed throughout for naming resource keys  
22. Validate markup for all web pages (validate HTML and CSS for syntax errors) to make sure it is compliant with the standards  
23. Application crash or unavailable pages should be redirected to error page  
24. Check text on all pages for spelling and grammatical errors  
25. Check numeric input fields with character input values. Proper validation message should appear  
26. Check for negative numbers if allowed for numeric fields  
27. Check amount fields with decimal number values  
28. Check functionality of buttons available on all pages  
29. User should not be able to submit page twice by pressing submit button in quick succession.  
30. Divide by zero errors should be handled for any calculations  
31. Input data with first and last position blank should be handled correctly

[**GUI**](http://www.softwaretestinghelp.com/gui-testing-on-smart-devices-%e2%80%93-testing-guidelines/) **and Usability Test Scenarios**

1. All fields on page (e.g. text box, radio options, dropdown lists) should be aligned properly  
2. Numeric values should be right justified unless specified otherwise  
3. Enough space should be provided between field labels, columns, rows, error messages etc.  
4. Scroll bar should be enabled only when necessary  
5. Font size, style and color for headline, description text, labels, infield data, and grid info should be standard as specified in SRS  
6. Description text box should be multi-line  
7. Disabled fields should be grayed out and user should not be able to set focus on these fields  
8. Upon click of any input text field, mouse arrow pointer should get changed to cursor  
9. User should not be able to type in drop down select lists  
10. Information filled by users should remain intact when there is error message on page submit. User should be able to submit the form again by correcting the errors  
11. Check if proper field labels are used in error messages  
12. Dropdown field values should be displayed in defined sort order  
13. Tab and Shift+Tab order should work properly  
14. Default radio options should be pre-selected on page load  
15. Field specific and page level help messages should be available  
16. Check if correct fields are highlighted in case of errors  
17. Check if dropdown list options are readable and not truncated due to field size limit  
18. All buttons on page should be accessible by keyboard shortcuts and user should be able to perform all operations using keyboard  
19. Check all pages for broken images  
20. Check all pages for broken links  
21. All pages should have title  
22. Confirmation messages should be displayed before performing any update or delete operation  
23. Hour glass should be displayed when application is busy  
24. Page text should be left justified  
25. User should be able to select only one radio option and any combination for check boxes.

**Test Scenarios for Filter Criteria**

1. User should be able to filter results using all parameters on the page  
2. Refine search functionality should load search page with all user selected search parameters  
3. When there is at least one filter criteria is required to perform search operation, make sure proper error message is displayed when user submits the page without selecting any filter criteria.  
4. When at least one filter criteria selection is not compulsory user should be able to submit page and default search criteria should get used to query results  
5. Proper validation messages should be displayed for invalid values for filter criteria

**Test Scenarios for Result Grid**

1. Page loading symbol should be displayed when it’s taking more than default time to load the result page  
2. Check if all search parameters are used to fetch data shown on result grid  
3. Total number of results should be displayed on result grid  
4. Search criteria used for searching should be displayed on result grid  
5. Result grid values should be sorted by default column.  
6. Sorted columns should be displayed with sorting icon  
7. Result grids should include all specified columns with correct values  
8. Ascending and descending sorting functionality should work for columns supported with data sorting  
9. Result grids should be displayed with proper column and row spacing  
10. Pagination should be enabled when there are more results than the default result count per page  
11. Check for Next, Previous, First and Last page pagination functionality  
12. Duplicate records should not be displayed in result grid  
13. Check if all columns are visible and horizontal scroll bar is enabled if necessary  
14. Check data for dynamic columns (columns whose values are calculated dynamically based on the other column values)  
15. For result grids showing reports check ‘Totals’ row and verify total for every column  
16. For result grids showing reports check ‘Totals’ row data when pagination is enabled and user navigates to next page  
17. Check if proper symbols are used for displaying column values e.g. % symbol should be displayed for percentage calculation  
18. Check result grid data if date range is enabled

**Test Scenarios for a Window**

1. Check if default window size is correct  
2. Check if child window size is correct  
3. Check if there is any field on page with default focus (in general, the focus should be set on first input field of the screen)  
4. Check if child windows are getting closed on closing parent/opener window  
5. If child window is opened, user should not be able to use or update any field on background or parent window  
6. Check window minimize, maximize and close functionality  
7. Check if window is re-sizable  
8. Check scroll bar functionality for parent and child windows  
9. Check cancel button functionality for child window

[**Database Testing**](http://www.softwaretestinghelp.com/database-testing-%e2%80%93-practical-tips-and-insight-on-how-to-test-database/) **Test Scenarios**

1. Check if correct data is getting saved in database upon successful page submit  
2. Check values for columns which are not accepting null values  
3. Check for data integrity. Data should be stored in single or multiple tables based on design  
4. Index names should be given as per the standards e.g. IND\_<Tablename>\_<ColumnName>  
5. Tables should have primary key column  
6. Table columns should have description information available (except for audit columns like created date, created by etc.)  
7. For every database add/update operation log should be added  
8. Required table indexes should be created  
9. Check if data is committed to database only when the operation is successfully completed  
10. Data should be rolled back in case of failed transactions  
11. Database name should be given as per the application type i.e. test, UAT, sandbox, live (though this is not a standard it is helpful for database maintenance)  
12. Database logical names should be given according to database name (again this is not standard but helpful for DB maintenance)  
13. Stored procedures should not be named with prefix “sp\_”  
14. Check is values for table audit columns (like createddate, createdby, updatedate, updatedby, isdeleted, deleteddate, deletedby etc.) are populated properly  
15. Check if input data is not truncated while saving. Field length shown to user on page and in database schema should be same  
16. Check numeric fields with minimum, maximum, and float values  
17. Check numeric fields with negative values (for both acceptance and non-acceptance)  
18. Check if radio button and dropdown list options are saved correctly in database  
19. Check if database fields are designed with correct data type and data length  
20. Check if all table constraints like Primary key, Foreign key etc. are implemented correctly  
21. Test stored procedures and triggers with sample input data  
22. Input field leading and trailing spaces should be truncated before committing data to database  
23. Null values should not be allowed for Primary key column

**Test Scenarios for Image Upload Functionality**

*(Also applicable for other file upload functionality)*  
1. Check for uploaded image path  
2. Check image upload and change functionality  
3. Check image upload functionality with image files of different extensions (e.g. JPEG, PNG, BMP etc.)  
4. Check image upload functionality with images having space or any other allowed special character in file name  
5. Check duplicate name image upload  
6. Check image upload with image size greater than the max allowed size. Proper error message should be displayed.  
7. Check image upload functionality with file types other than images (e.g. txt, doc, pdf, exe etc.). Proper error message should be displayed  
8. Check if images of specified height and width (if defined) are accepted otherwise rejected  
9. Image upload progress bar should appear for large size images  
10. Check if cancel button functionality is working in between upload process  
11. Check if file selection dialog shows only supported files listed  
12. Check multiple images upload functionality  
13. Check image quality after upload. Image quality should not be changed after upload  
14. Check if user is able to use/view the uploaded images

**Test Scenarios for Sending Emails**

*(Test cases for composing or validating emails are not included)*  
*(Make sure to use dummy email addresses before executing email related tests)*  
1. Email template should use standard CSS for all emails  
2. Email addresses should be validated before sending emails  
3. Special characters in email body template should be handled properly  
4. Language specific characters (e.g. Russian, Chinese or German language characters) should be handled properly in email body template  
5. Email subject should not be blank  
6. Placeholder fields used in email template should be replaced with actual values e.g. {Firstname} {Lastname} should be replaced with individuals first and last name properly for all recipients  
7. If reports with dynamic values are included in email body, report data should be calculated correctly  
8. Email sender name should not be blank  
9. Emails should be checked in different email clients like Outlook, Gmail, Hotmail, Yahoo! mail etc.  
10. Check send email functionality using TO, CC and BCC fields  
11. Check plain text emails  
12. Check HTML format emails  
13. Check email header and footer for company logo, privacy policy and other links  
14. Check emails with attachments  
15. Check send email functionality to single, multiple or distribution list recipients  
16. Check if reply to email address is correct  
17. Check sending high volume of emails

**Test Scenarios for Excel Export Functionality**

1. File should get exported in proper file extension  
2. File name for the exported Excel file should be as per the standards e.g. if file name is using timestamp, it should get replaced properly with actual timestamp at the time of exporting the file  
3. Check for date format if exported Excel file contains date columns  
4. Check number formatting for numeric or currency values. Formatting should be same as shown on page  
5. Exported file should have columns with proper column names  
6. Default page sorting should be carried in exported file as well  
7. Excel file data should be formatted properly with header and footer text, date, page numbers etc. values for all pages  
8. Check if data displayed on page and exported Excel file is same  
9. Check export functionality when pagination is enabled  
10. Check if export button is showing proper icon according to exported file type e.g. Excel file icon for xls files  
11. Check export functionality for files with very large size  
12. Check export functionality for pages containing special characters. Check if these special characters are exported properly in Excel file

**Performance Testing Test Scenarios**

1. Check if page load time is within acceptable range  
2. Check page load on slow connections  
3. Check response time for any action under light, normal, moderate and heavy load conditions  
4. Check performance of database stored procedures and triggers  
5. Check database query execution time  
6. Check for load testing of application  
7. Check for stress testing of application  
8. Check CPU and memory usage under peak load condition

[**Security Testing**](http://www.softwaretestinghelp.com/security-testing-of-web-applications/) **Test Scenarios**

1. Check for SQL injection attacks  
2. Secure pages should use HTTPS protocol  
3. Page crash should not reveal application or server info. Error page should be displayed for this  
4. Escape special characters in input  
5. Error messages should not reveal any sensitive information  
6. All credentials should be transferred over an encrypted channel  
7. Test password security and password policy enforcement  
8. Check application logout functionality  
9. Check for Brute Force Attacks  
10. Cookie information should be stored in encrypted format only  
11. Check session cookie duration and session termination after timeout or logout  
11. Session tokens should be transmitted over secured channel  
13. Password should not be stored in cookies  
14. Test for Denial of Service attacks  
15. Test for memory leakage  
16. Test unauthorized application access by manipulating variable values in browser address bar  
17. Test file extension handing so that exe files are not uploaded and executed on server  
18. Sensitive fields like passwords and credit card information should not have auto complete enabled  
19. File upload functionality should use file type restrictions and also anti-virus for scanning uploaded files  
20. Check if directory listing is prohibited  
21. Password and other sensitive fields should be masked while typing  
22. Check if forgot password functionality is secured with features like temporary password expiry after specified hours and security question is asked before changing or requesting new password  
23. Verify CAPTCHA functionality  
24. Check if important events are logged in log files  
25. Check if access privileges are implemented correctly

<http://www.softwaretestinghelp.com/sample-test-cases-testing-web-desktop-applications/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BRD stands for Business Requirement Document.

**For us testers,** [**test cases**](http://www.softwaretestinghelp.com/how-to-write-effective-test-cases-test-cases-procedures-and-definitions/) **are the real deal** – the stuff that we spend most of our times around. We create them, review them, execute them, maintain them, automate them- well, you get the picture. No matter how experienced we are and what role we play in a project – we would still work with test cases.

QA plan : <http://readyset.tigris.org/nonav/templates/qa-plan.html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing Tutorials

Application testing is when you have software, you're testing it for bugs or problems. Basically you're trying to get it to fail any way you can. You try finding the weak spots so you can fix them, so when the application is sent out it is free of bugs.

**Introduction to Software Testing**

 **Software testing is a process used to identify the correctness, completeness, and quality of developed computer software.** It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users.

 In simple words, **software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is defect free**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Summary of the Seven Testing Principles**   |  |  | | --- | --- | | Principle 1 | Testing shows presence of defects | | Principle 2 | Exhaustive testing is impossible | | Principle 3 | Early Testing | | Principle 4 | Defect Clustering | | Principle 5 | Pesticide Paradox | | Principle 6 | Testing is context dependent | | Principle 7 | Absence of errors - fallacy | |

One of the  Software Testing Fundamental is "**100% Automation is not possible**".

<http://guru99.com/software-testing-life-cycle.html>

[Test plan](http://guru99.com/test-plan.html) /strategy document🡪 <http://guru99.com/test-plan.html>

Test Data🡪 http://guru99.com/software-testing-test-data.html

## Software Testing Life Cycle STLC🡪 http://guru99.com/software-testing-life-cycle.html

|  |  |
| --- | --- |
|  | **Unit testing**  Unit testing of software applications is done during the development (coding) of an application.  The objective of unit testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure  The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing is usually performed by the developer. |

**Integration Testing**

**for testers**

In Integration Testing, individual software modules are integrated logically and tested as a group.

A typical software project consists of multiple software modules, coded by different programmers.  Integration testing focuses on checking data communication amongst these modules.

Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes 'Thread Testing'.

Integration Test case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**.

<http://guru99.com/integration-testing.html>

**system testing**

In the world of software testing, system testing is the testing of a complete and fully integrated software product.

Usually software is only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware systems. **System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system**.

## The Software Testing Hierarchy

As with almost any technical process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

* **Unit testing -** testing performed on each module or block of code during development. Unit testing is normally done by the programmer who writes the code.
* **Integration testing -** testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
* **System testing -** testing done by a professional testing agent on the completed software product before it is introduced to the market.
* **Acceptance testing -** beta testing of the product done by the actual end users.

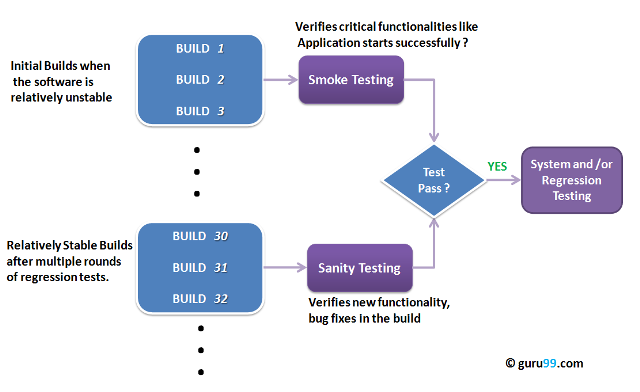
## Types of System Testing

There are more than 50 types of System Testing. For an exhaustive list of software testing types click [here](http://guru99.com/types-of-software-testing.html). Below we have listed types of system testing a large software development company would typically use

1. **Usability Testing -** Usability testing mainly focuses on the user's-ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. **Load Testing -** Load testing is necessary to know that a software solution will perform under real life loads.
3. **Regression Testing** - Regression testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. **Recovery Testing -** Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. **Migration Testing -** Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. **Functional Testing -** Also known as functional completeness testing, functional testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. **Hardware/Software Testing -** IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

## 100 Types of Software Testing -🡪 http://guru99.com/types-of-software-testing.html

## Smoke and Sanity Testing



Smoke Testing is performed after software build to **ascertain that the critical functionalities of the program is working fine**.It is executed **"before"** any detailed functional or regression tests are executed on the software build.The **purpose is to reject a badly broken application**, so that the QA team does not waste time installing and testing the software application.

After receiving a **software build, with minor changes in code, or functionality, Sanity testing is performed to ascertain that the bugs have been fixed and no further issues are introduced due to these changes**.The goal is to determine that the proposed functionality works roughly as expected. **If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing**.

**regression testing**

The purpose of regression testing is to confirm that a  recent program or code change has not adversely affected existing features.

Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It  ensures that old code still works once  the new code changes are done.

|  |
| --- |
| Regression Testing is required when there is a   * Change in requirements and code is modified according to the requirement * New feature is added to the software * Defect fixing * Performance issue fix |

**Regression Testing Techniques**

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly . Therefore , Regression Testing becomes necessary.

**Selenium:**This is an open source tool used for automating web applications. Selenium can be used for browser based regression testing.

<http://guru99.com/regression-testing.html>

End-to-End Testing

Unlike System Testing, End-to-End Testing not only validates the software system under test but also  checks it's integration with external interfaces. Hence, the name "**End-to-End**". The purpose of End-to-End Testing is to exercise a complete production-like scenario. Along with the software system, it also validates batch/data processing from other upstream/downstream systems.

End to End Testing is usually executed after functional and system testing. It uses actual production like data and test environment to simulate real-time settings. End-to-End testing is also called Chain Testing

|  |
| --- |
| Positive and Negative Testing  **Software testing** is process of verifying and validating the software or application and checks whether it is working as expected. The intent is to find defects and improve the product quality. There are two ways to test the software , viz,  Positive Testing and Negative Testing. |

**Positive testing** can be performed on the system by providing the **valid data** **as input.** It checks whether an application behaves as expected with the positive input. . This is to test to check the application that does what it is  supposed to do so

**Negative Testing** can be performed on the system by providing **invalid data as input**. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do so.

Adhoc testing

Adhoc testing is an informal testing type with an aim to break the system. This testing is usually an unplanned activity . It does not follow any test design techniques to create test cases. In fact is does not create test cases altogether! This testing is primarily performed if the knowledge of testers in the system under test is very high. Testers randomly test the application without any test cases or any business requirement document.

Adhoc Testing does not follow any structured way of testing and it is randomly done on any part of application. Main aim of this testing is to find defects by random checking. Adhoc testing can be achieved with the testing technique called **Error Guessing.** Error guessing can be done by the people having enough experience on the system to "guess" the most likely source of errors.

Adhoc testing can be performed when there is limited time to do elaborative testing. Usually adhoc testing is performed after the formal test execution. And if time permits, adhoc testing can be done on the system).Adhoc testing will be effective only if the tester is knowledgeable of the System Under Test.

## Types of adhoc testing

There are different types of Adhoc testing and they are listed as below:

|  |  |
| --- | --- |
| **Buddy Testing**  http://cdn.guru99.com/images/Adhoc2.png | Two buddies mutually work on identifying defects in the same module. Mostly one buddy will be from development team and another person will be from testing team. Buddy testing helps the testers develop better test cases and development team can also make design changes early. This testing usually happens after unit testing completion. |
| **Pair testing**  http://cdn.guru99.com/images/Adhoc3.png | Two testers are assigned modules, share ideas and work on the same machines to find defects. One person can execute the tests and another person can take notes on the findings. Roles of the persons can be a tester and scriber during testing.  **Comparison Buddy and Pair Testing**  Buddy testing is combination of unit and system testing together with developers and testers but Pair testing is done only with the testers with different knowledge levels.(Experienced and non-experienced to share their ideas and views) |
| **Monkey Testing**  http://cdn.guru99.com/images/Adhoc4.png | Randomly test the product or application without test cases **with a goal to**break the system. |