|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | | | |
|  | | | |
| **QA Glossary** |  |

**A**

**AcceptanceTesting**

(Alsocalled "certificationtesting")

A quick test to determine if software is ready for release

Acceptance testing follows regression testing.

As a rule, test documentation for acceptance testing comes in the form of a checklist.

**Actual Result**

*In general:* whatever happens in reality as a consequence of something

*Regarding the software:* actual output

**Ad Hoc Testing**

Testing done without any preparation

Doing ad hoc testing, the tester just follows his or her heart to try to find bugs.

**Alpha Testing**

Testing done before a release

When you hear "alpha testing," it refers to **when** the testing is done, not **how** the testing was done. As a rule, alpha testing is done inside the company. In some cases, alpha testing is outsourced to other companies.

Also see *Beta Testing*

**Also Notify** (in BTS)

The list of persons who should get emails containing:

- Notification that the bug has been filed

- Updates about changes to the bug

**Application Core**

The heart of the system responsible for processing

Also see *Web Site Architecture*

**Application Version**

The version of the application formatted according to the convention accepted at the company

For example, at ShareLane we have this convention:

*<major release number>.<minor release number>-<build number>/<DB version number>*

e.g., 1.0-23/34

**ASCII**

"Character encoding based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text." (Wikipedia)

**Assigned by** (in BTS)

Alias of the person who assigned the bug to its current owner

**Assigned to** (in BTS)

Alias of the current bug owner

**Attachment** (in BTS)

File (usually an image) uploaded as an illustration to the bug

**Automated Testing**

Testing completely done by running test automation tools.

Simple to perform yet valuable, automated testing can be done with link checkers; e.g., Xenu Link Sleuth (you can download it from QATutor.com).

Also see *Manual Testing, Semi-Automated Testing, Test Automation*

**Automation Script**

(In the context of this course) Test automation written for the regression testing of

- components

- end-to-end flows

Also see *Helpers*

**B**

**Back End**

Software and data hidden from the user; e.g., Web server, application core, the DB, log files, etc.

Compared to a car, the back end pieces are items like the engine and the electrical circuit of the car.

Also see *Front End*

**Beta Testing**

Testing done after a release to a narrow audience

The value of beta testing is allowing **some** representatives of our target audience to try the product and give us feedback before we release it in the open.

Also see *Alpha Testing*

**Black Box Testing**

Manual or semi-automated testing that occurs when:

- The tester usually has no idea about the internals of the back end

- Ideas for testing come from expected patterns of user behavior

Also see *Grey Box Testing; White Box Testing*

**Black Box Testing Methodology**

A set of black box testing techniques and approaches

**Boundary Testing**

Testing using the *Boundary Values* technique

**Boundary Values**

Two meanings:

1. Extreme border values of the equivalent class

2. Black box testing technique that involves the boundary values of the equivalent classes

**Brain Positioning**

The most vital fundamental concepts and attitudes regarding the subject of study

**Bug**

Depending on the context:

1. The deviation of an actual result from an expected result

2. A problem report with the "Bug" type inside the Bug Tracking System

Also see, *Software Bug*

**Bug Attributes** (in BTS)

Attributes of bug report in BTS

-ID

-Summary

-Description

-Attachment

-Submitted by

-Date

-Assigned to

-Assigned by

-Verifier

-Component

-Found on

-Version

-Build

-DB

-Comments

-Severity

-Priority

-Also notify

-Change history

-Type

-Status -Resolution

**Bug Fix Verification**

A 2-step process:

1. Verification that the bug was really fixed

2. Checking whether the bug fix has introduced other bugs

The bug can be closed right after Step 1 if the bug is no longer reproducible.

**Bug Owner**(in BTS)

The person responsible for the next step in bug closure

**Bug Postmortem**

A meeting to discuss why bugs that went to production weren't caught during testing

This meeting should be about improvements, not about blaming.

**Bug Regression**

See *Bug Fix Verification*

**Bug Tracking Procedure (BTP)**

The set of rules about how bug tracking should function from the moment a bug is found and filed (or re-opened) to the moment when a bug is closed

**Bug Tracking System (BTS)**

Infrastructure that enables testers to

- create,

- store,

- view and

- modify

information about bugs

**Bugzilla**

Free, popular, and easy to use BTS software

**Build**

A sub-version of the specific release

**Build** (in BTS)

The build number of the application version where the bug was found

If the application version is 1.0-23/34, the build in the BTS must be **23.**

**Build ID**

Unique identifier of sub-version of the software for concrete release.

For example:

1.0-23

**Build Number**

Value that shows how many builds have been created for concrete release.

Build numbering starts with 1 and increases in increments of 1 every time a new build is created

For example:

1.0-23 means that there were 23 new builds for release 1.0.

**C**

**Change History** (in BTS)

A log of the changes that occur with a bug

The Change History usually includes:

- Date and exact time (to the second) of the bug filing/change

- Alias of the person who filed a bug/made a change

- Fact that the bug was filed, or what was changed and how it was changed

**Checkbox** (in HTML)

Used for situations where there is a need to select 1 or more options

Unlike radio buttons, there are no relationships between checkboxes.

For example:

Making a sandwich

**Comments** (in BTS)

This attribute has 2 main usages: Bread Lettuce Cheese  
 - Comments about the bug itself; e.g., provide more info about the steps to reproduce

- Comments about bug-related actions; e.g., when a developer reassigns a bug to another developer

**Compatibility Testing**

Checking the compatibility of **our** software with the expected hardware and/or software on a user's computer

**Compiler**

Compilers and interpreters are special programs that translate code typed by humans (a=2+3) into the language understood by computers (0011010100101010101001111)

The difference between compiler and interpreter is this:

- A compiler produces an executable file, but doesn't execute our code. Therefore, text files written in C++ must be compiled to become executable (launchable) programs.

- An interpreter doesn't produce an executable file but immediately executes our code. Therefore, text files written in Python are already executable.

**Component** (in BTS)

The component where a bug was found - e.g., Checkout

**Component Testing**

Functional testing of a logical component

Also see *Integration Testing; System Testing*

**Condition**

A preset or the environment

For example, we can try to register using:

- a brand new email address

OR

- an email address that has already been used for registration

"Email is not in database" and "Email is in database" are two conditions.

**CVS**

Free version control software

**D**

**Data**

A piece (or pieces) of information

Data usually comes in the form of text, a file, a value of a variable, or a value inside a DB record.

For each concrete situation, data is either:

- **Valid**

OR

- **Invalid**

For each concrete situation, Null input can be considered as either valid or invalid data.

**Data Integrity**

"Correctness, completeness, wholeness, soundness, and compliance with the intention of the creators of the data" (definition taken from TechWeb.com)

If data integrity is seriously compromised in the test environment, testing might not make much sense until data integrity is reestablished.

**Database (DB)**

Conceptually, the DB is a set of virtual containers required to organize and store data

The most popular DBs used in the Internet software industry are MySQL (free) and Oracle (not free).

Also see *Web Site Architecture*

**Database Administrator (DBA)**

A professional who manages databases

**Date** (in BTS)

Date and time when a bug was filed

**DB** (in BTS)

The DB version of the environment where a bug was found

If the application version is 1.0-23/34, the DB in the BTS must be **34.**

**Debugging**

The activity of a programmer to identify a buggy piece of code and fix it

**Deprecated Code**

A piece of code that has not been removed from the software, but should not be used

Often, code is not removed but deprecated to make a **safe transition** from an old (deprecated) code to a new code. Deprecated code serves as a **backup** in case something goes wrong with the new code. Once the new code has been there for a while and there have been many opportunities to analyze the consequences of removing the deprecated code, we might want to get rid of it.

**Description** (in BTS)

A detailed description of the bug

Here is the recommended format for Description:

**Description:**<provide information about what happens>

**Steps to reproduce:**<provide steps to reproduce the bug>

**Bug:**<state what's wrong>

**Expected:**<state what's expected>

Also see *Summary*

**Developer**

A professional who writes software code

**Development Environment**

(Also called "playground" or "sandbox")

The software/hardware combo where a developer writes and tests his or her code

**Dirty List - White List**

A black box testing technique that consists of 2 parts:

Part 1: Brainstorming (black list)

Part 2: Selection of items that came up during Part 1 (white list)

**Drop-Down Menu** (in HTML)

A list of values to choose from

Forexample:



**E**

**Emergency Bug Fix (EBF)**

A situation when a P1 bug is found in production and it's necessary to create a patch release ASAP

**EBF procedure**

Set of rules on how to proceed in case of EBF or EFR

**Emergency Feature Request (EFR)**

A situation when a certain feature must be released ASAP; e.g., in the case of a court ruling or to comply with a new law

For example:

Our competitor has won a patent case, and we have to change some piece of our software ASAP to make it work in a different way.

**End-to-End Testing**

See *System Testing*

**Entry Criteria**

Certain conditions that allow to begin something

For example:

To make a phone call, you must have a working phone, a connection, and the phone number of the recipient. So we can say that the entry criteria for a phone call includes 3 conditions:

- a working phone is available

- a connection is available

- the phone number is known

Also see *Exit Criteria*

**Equivalent Classes**

A set of inputs that are treated by software the same way under certain conditions

In other words, under certain conditions, the software must apply the same logic to each element of the equivalent class.

In some cases, the equivalent class can consist of only one element. Also see *Boundary Values*

**Error Message**

A message that provides information about error(s)

An error message is an important measure that:

- **Guides users in case of mistakes**

An error message is usually delivered via a Web page by a code that is specifically written for handling user errors.

- **Gives debugging info to developers**

An error message is usually provided by an interpreter (or a compiler), or by some logging mechanism: e.g., the Apache Web server records errors in a special error log.

**Error Handling**

1. How the system responds to errors made by users

An example is the way the system responds if a Web form is submitted with invalid data in a required field.

2. How the system reacts to errors that happen when the software is running

For example, this error message: Test Portal>More Stuff>Python Errors>register\_with\_error.py provides info that the file register\_with\_error.py calls the undefined function get\_firstpage().

**Exit Criteria**

Certain conditions that allow something to be considered finished.

For example, lunch at a restaurant is finished when the bill is paid. The exit criteria for a meal at a restaurant is:

- The bill is paid

Also see *Entry Criteria*

**Expected Pattern of User Behavior**

Scenarios that we expect will be (OR are already) taking place as users use our software

**Expected Result**

*In general:* whatever is expected to happen in reality as a consequence of something

*Regarding the software:* expected output

**Sources of expected results**:

- Specification

- Life experience

- Common sense

- Communication

- Standards

- Statistics

- Valuable opinion

- Others sources

The real challenge is to find expected results that serve as a **true indicator** of whether the software works or not.

**Exploratory Testing**

Exploration for the purpose of finding bugs

Also see *Ad hoc testing*

**Exploring**

Browsing through software UI to understand how things work.

**F**

**Feature**

Depending on the context, the term "feature" means:

- the ability to accomplish a specific task (i.e., *functionality*)

OR

- a particular characteristic of the software

**Feature Release**

see*Minor Release*

**Flow**

See *Scenario*

**Flowchart**

See *Process Flowchart*

**Formal (Documented) Testing**

(In context of this course) Test execution with help of test documentation; e.g., test cases

**Found On** (in BTS)

The environment where a bug was found

For example: main.sharelane.com.

**Front End**

The interface that customers can see and use; e.g., text, images, buttons, or links

Compared to a car, the front end pieces are items like the steering wheel and the dashboard.

Also see *Back End*

**Functional Testing**

Testing with the purpose of finding bugs in functionalities of the software.

**Functionality**

The ability to accomplish a certain task

For example, the functionality (ability) of a bottle opener is to open bottles.

Alsosee *Feature*

**G**

**Grey Box testing**

Testing that combines the elements of black and white box testing:

- On the one hand, the tester uses black box methodology to come up with test scenarios

- On the other hand, the tester possesses some knowledge about the back end, AND he or she actively uses that knowledge

**H**

**Helpers**

(In context of this course) a type of test automation needed to automate manual repetitive tasks

The most popular example of a helper is the utility for the automated creation of new user accounts; e.g., the Account Creator used at ShareLane.

**I**

**ID** (in BTS)

The unique ID of the bug

(HTML) **Image**

A graphic file on a Web page

Forexample:



**Input**

Depending on the context:

- data

- scenario

Also see *Output*

**Integration Testing**

Functional testing of the interaction between two or more integrated components

Also see *Component Testing; System Testing*

**Internet**

"The global system of interconnected computer networks" (shortened definition from Wikipedia)

**Interpreter**

See *Compiler*

**Intranet**

Local network within a company

**Invalid Data**

Data that should NOT be able to assist in accomplishing some task; e.g., registration

In the case of registration at ShareLane, a ZIP code must contain 5 digits. All other inputs are invalid data.

**J**

**Job Security**

Ability to find a job in any economic situation. Job security is about your professionalism and not about your company

**L**

**Legacy <something>**

The word "legacy" is usually used in the terms "legacy feature" or "legacy user". It means "existing". For example, if production has a feature called "Shopping Cart", then the "Shopping Cart" is a legacy feature.

**Link** (in HTML)

Navigation element

The link can point to

- a URL

- a position on a Web page

Here are the typical targets for links:

1. File - *e.g.,* HTML file (Web page), PDF file, TXT file, Python file, etc.

2. Anchor on *the same* Web page

3. Anchor on *adifferent* Web page

4. mailto value

For example:

[**Sign up**](http://www.sharelane.com/cgi-bin/register.py)

**Linked Image** (in HTML)

A link presented in the form of an image

Forexample:

[](http://www.sharelane.com/cgi-bin/show_book.py?book_id=8)

**Load/Performance Testing**

A set of testing techniques designed to load the system or its components and then measure how the system or its components react

The usual purpose of load/ performance testing is to find a bottleneck; e.g., a part of the system or its components that slows down response time.

**Localization Testing**

The testing used to find bugs in the **adaptation** of the software by users from different countries

For example, if our Web site was created for an English-speaking audience and we want to localize it for a Japanese-speaking audience, we'll have to determine whether Kanji symbols can be used to create a username.

**Log File**

A file that keeps track of some activity

There are 3 most common actions regarding data in log files:

1. Read (lines are read by humans or a program)

2. Append (new lines are added under old lines, if any)

3. Write (all old lines if any are purged and new lines are added)

**Logical Bug**

A bug in how the software **processes** information

Also see *Syntax bug; UI bug*

**M**

**Major Release**

Major (or milestone) release that happens at the release stage of the SDLC, after the testing and bug fixes stage is over

The version of a major release is presented as an integer: **7**.0

**Manual Testing**

Testing that doesn't require any automated tools

This term is usually applied towards black/grey box testing.

Also see *Automated Testing, Semi-Automated Testing*

**Minor Release**

A release that takes place between major releases

A minor release can have one of three variants:

- Feature release

- Patch release

- Mixed release

A FEATURE RELEASE takes place when we need to:

- Add new features

and/or

- Modify/remove existing features

A PATCH RELEASE takes place when the code in production has a bug(s). In this case, we simply release the fixed code.

A MIXED RELEASE is minor release with both feature related changes and bug fixes.

The version of a minor release is presented as a number after the decimal point and incremented by one after each further minor release: 7.**1**

**Mixed Minor Release**

See*Minor Release*

**N**

**Negative Testing**

Testing that checks situations involving:

- **User error**

and/or

- **System failure**

Also see *Positive Testing*

**New Feature Testing (NFT)**

The first stage of test execution where new and/or changed features are tested

Also see *Regression Testing*

**Null Input**

No data is provided

For example, if we press "Continue" on the first page of the ShareLane registration without entering **anything** into the "ZIP code" field, this is null input.

**O**

**Output**

Result produced by the software in response to input

**P**

**Password Input Box** (in HTML)

Special 1-line text box where the input is masked by asterisks or dots as the user types text

Example:

Password: 

**Patch Release**

See*Minor Release*

**PjM or Project Manager**

A professional who manages projects

"They have the responsibility of the planning, execution, and closing of any project" (Wikipedia).

As a rule, in a start-up environment, the PjM's role is assigned to the PM.

**PM or Product Manager**

A professional who manages products

"A product manager researches, selects, develops, and places a company's products" (Wikipedia).

One of the main deliverables expected from a PM are well-written specs.

**Positive Testing**

Testing that checks situations where:

- The software is used in a normal, error-free way and/or

- The system is assumed to be sound

Usage in a "normal, error-free way" can be defined as a scenario that accomplishes certain tasks needed to provide some value to a user. For example, registration is needed to create an account. So,

- Normal usage correctly completes all steps of the registration needed to create new account.

- Abnormal usage in this case would be submitting a Web form during registration where certain fields (e.g., ZIP code) have invalid data.

Also see *Negative Testing*

**Postmortem**

See *Bug Postmortem*

**Priority** (in BTS)

The magnitude of a bug's impact on the company's business

**Prod**

See *Production*

**Production**

A Web site available to our users

The opposite of prod are the development and test environments where the software is being developed and tested.

**Production environment**

See *Production*

**Programmer**

See *Developer*

**Process Flowchart**

A graphical presentation of the process

**Q**

**Quality**

State or characteristic attributed to something (functionality, code, overall product, etc.) based on degree of match between that "something" and someone's expectations about it.

Example #1: A tester says: "The quality of the checkout flow is good, because we fixed all the bugs." (The expectation is: "Good software is bug-free software.")

Example #2: A user says: "The quality of the checkout flow sucks, because the UI is very misleading." (The expectation is: "Software should have an easy-to-use interface.")

**Quality Assurance (QA)**

The set of activities targeted at bug prevention through process improvement

**QA Engineer**

*In theory:* A professional specializing purely in process improvement

*In reality:* This term is used interchangeably with "test engineer" and "tester".

**R**

**Radio Button** (in HTML)

The element of a Web form that allows for the selection of one, and only one, value from a group with **the same** name (within the same Web form)

For example:

Morning plan

Go fishing  
Play with kids  
Repair dishwasher

**Regression Testing (RT)**

Checking to determine if legacy features are broken because of a particular change in the software; e.g.:

- The introduction of new features

- A bug fix

There are 2 reasons for regression testing:

1. It's often extremely difficult for a programmer to figure out how a change in one part of the software will echo in other parts of the software

AND**what's even worse**

2. Programmers sometimes change software **without even trying** to figure out if their changes might break something else

As a stage of test execution, regression testing comes after new feature testing.

Also see *Bug Verification*

**Release Engineer (RE)**

The person responsible for creating the release engineering infrastructure and for pushing code to various environments; e.g., test environments or production

**Required Field**

Web page element that should to be filled with valid data (e.g., text box) or which value must be selected (e.g., value in drop-down menu) to be able to proceed to the next Web page.

For example, in the "Email" field, valid data must have one and only one "@" character.

Null input is always considered to be invalid for the required field.

**Reset Button** (in HTML)

Pressing this button restores the values inside a Web form back to their defaults

For example (type something and press the button):

Начало формы



Конец формы

**Resolution** (in BTS)

A stage of a bug's life

Explanations below are given with the assumption that bug report has Type "Bug":

**>Reported**: A bug was filed, but the developer to fix it has not been assigned.

**>Assigned**: Assigned developer must start bug investigation.

**>Fix in progress**: The developer is fixing the bug.

**>Fixed**: The bug was fixed, but the bug fix hasn't been verified yet.

**>Fix is verified**: The bug fix has been verified.

**>Verification failed**: The bug fix verification failed; i.e., bug is reproducible after the bug fix.

**>Cannot reproduce**: The developer cannot reproduce the bug.

**>Duplicate**: The bug is a duplicate of another bug.

**>Not a bug**: The bug is not considered to represent a problem (i.e., a deviation of actual from expected).

**>3rd party bug**: The bug is in 3rd party software.

**>No longer applicable**: The bug doesn't have any meaning anymore.

**Risk Analysis**

(In the context of this Course) A black box testing technique based on evaluation of data or expectations with the purpose of setting priorities

**Rollback**

Action(s) to undo unwanted changes.

**S**

**Scenario**

A combination of actions and data applied to software under certain conditions.

The purpose of a scenario is to bring test execution to the point where an actual result can be retrieved and compared with an expected result.

In the example below, the verbs "Go", "Type" and "Click" point to **actions**. **Data**is presented by the word "expectations". If scenario assumes that book is in DB, then **condition**is:*DB has data about book with word "expectations" in its title*.

1. Go to www.sharelane.com.

2. Type word "expectations" into the text field "Search".

3. Click the "Search" button.

**Security Testing**

Testing for protection against security breaches

**Semi-Automated Testing**

Manual testing done with partial usage of the test automation, usually in form of helpers

Also see *Automated Testing, Manual Testing*

**Severity** (in BTS)

The magnitude of a bug's impact on the software

**Smoke Test**

(Also called "sanity test" or "confidence test") A check to determine if the software is testable

During a smoke test, we check the main flows of the main features.

**Software Bug**

"An error, flaw, mistake, failure, fault, or 'undocumented feature' in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result)" (Wikipedia)

**Software Development Life Cycle (SDLC)**

A way to get

- from an idea about a *desired software*

- to the release of the *actual software* and its maintenance

**Software Testing**

A set of activities and processes primarily targeted to find AND address software bugs

**Spec**

The instruction on how software should work and/or look

**Start-up**

A young company that usually has a short and eventful life

If:

- your share in company is 0.1% or more

- you eat free pepperoni pizza at least once a week

- you imagine yourself to be filthy rich within 4 years

then it's likely that you work for a start-up. Some start-ups like Google, Netscape and YouTube did make their early employees multimillionaires.

**Status** (in BTS)

The state of a bug:

- Open

- Closed

- Re-open

**Structured Query Language (SQL)**

SQL is a language to communicate with DB

"SQL is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management" (Wikipedia).

**Submitted by** (in BTS)

The alias of the person who filed a bug

**Submit Button** (in HTML)

Button used to submit Web form to the Web server

Example:

Начало формы



Конец формы

**Summary** (in BTS)

A quick synopsis of the bug

Also see *Description*

**Syntax Bug**

A bug in the syntax of the software code

Also see *Logical bug; UI bug*

**System Testing**

Functional testing of a logically complete path

This term is usually applied to situations where two or more integrated components are involved.

Also see *Component Testing; Integration Testing*

**T**

**Test Automation (TA)**

*In general:* A myriad of different tools and techniques for a myriad of different purposes in software testing: code analysis, link checking, load/performance testing, code coverage, unit testing - this list goes on and on.

*In context of Lecture 10:* Test automation for regression testing.

Also see *Helpers; Automation Scripts*

**Test Case**

*Conceptually:* an idea about verifying something - e.g., if the vacuum cleaner works, or if your taxi arrived to take you to the airport.

The essential part of a test case is the expected result.

*As a document:* a container with

- The scenario on one hand

and

- The expected result on the other hand

A scenario is required to bring the test case executor (the person or program) to the actual result. The expected result serves as a benchmark to compare with the actual result and conclude if they match or not. The purpose of a test case is to find a bug.

**Test Case Attributes**

Useful parts of the test case that assist with:

- **Test case execution**; e.g., an IDEA that clarifies what we are checking by using that test case

- **Test case management**; e.g., SETUP AND ADDITIONAL INFO that can contain data to make test cases more maintainable

The most common test case attributes are:

- Unique ID

- Priority

- IDEA

- SETUP AND ADDITIONAL INFO

- Revision History

**Test Coverage**

Depending on the context, this means one of the following:

1. The coverage of possible scenarios

2. Test case execution coverage

**Test Cycle**

A 2-stage process:

Stage 1: Test preps

Stage 2: Test execution

**Test Engineer**

A professional specializing in software testing

**Test Environment**

The software/hardware combo where software is tested before being released to production

**Test Execution**

The second stage of the Test cycle

**Test Plan**

The master document containing information about activities regarding the testing of the certain features (or other possible subjects of testing - e.g., how the system handles load)

**Test Preps**

The first stage of the test cycle

**Test Suite**

A collection of test cases, usually dedicated to the same spec or the same feature

**Test Tables**

A black box testing technique that involves creating tables with inputs and/or conditions, and then combining those tables into test scenarios

**Testability**

Whether it makes sense to start testing

For example:

If a major flow (e.g., login) is not functioning, testing is blocked; therefore, we can say that the software is not testable.

**Tester**

See *Test Engineer*

**Testing**

See *Software Testing*

**Text** (in HTML)

The text on a Web page

For example:

I'm text.

**Text Area** (in HTML)

A multiline input field for text on a Web page

Forexample:



**Text Box** (in HTML)

A 1-line input field for text on a Web page

Forexample:

Начало формы



Конец формы

**The Cycle**

see*Software Development Life Cycle*

**Type** (in BTS)

The kind of bug report:

- Bug

- Feature request

**U**

**UI Bug**

A bug in how software **presents** information

Also see *Logical Bug; Syntax bug*

**UI Testing**

The type of testing needed to find bugs in the user interface.

Also see *UI Bug*

**Unit Testing**

"Method of testing that verifies the individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class." (Wikipedia)

Unit testing is usually performed by the programmer against his or her own freshly baked code

**UNIX Timestamp**

"The number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds" (Wikipedia)

**URL**

The unique address of the resource (usually a file) on the network

**Usability Testing**

The evaluation of the user's experience when he or she uses our software

**Use Case**

A description of:

- How software will be (or is) used

- How software must respond to certain scenarios

**User Interface (UI)**

The part of the software that enables a user to use the software

The typical elements of a UI for Web projects are text, image, link, text field, button, etc.

Software usage comes in many shapes and forms - e.g., from watching videos on YouTube to transferring money using PayPal.

**V**

**Valid Data**

Data that should be able to assist in accomplishing some task; e.g., registration

In the case of registration at ShareLane, valid data for the ZIP code is 5 digits. Any other data is invalid.

**Verifier** (in BTS)

The alias of the person who must verify the bug after it was fixed

**Version** (in BTS)

The version of the environment where the bug was found

If the application version is 1.0-23/34, the version in the BTS must be **1.0**

**W**

**Web (Word Wide Web)**

"A computer network consisting of a collection of Internet sites that offer text and graphics and sound and animation resources through the hypertext transfer protocol" (definition taken from wordnet.princeton.edu)

**Web Form** (in HTML)

"A Web form on a Web page allows a user to enter data that is, typically, sent to a server for processing and to mimic the usage of paper forms." (Wikipedia).

**Web Server**

"A computer program that is responsible for accepting HTTP requests from Web clients, which are known as Web browsers, and serving them HTTP responses along with optional data contents, which usually are Web pages such as HTML documents and linked objects (images, etc.)" (Wikipedia).

**Web Site**

(In the context of this course) An Internet project with the UI available to users via the Web

Also see *Web Site Architecture*

**Web Site Architecture**

The typical Web site architecture looks like this:

- Web server (HTTP handling)

- Application core (processing)

- Database (storage)

**White Box Testing**

(Also called "glass box testing", "clear box testing", and "open box testing")

The number of testing techniques that require a comprehensive understanding of the software code

For example:

A programmer can perform white box testing by comparing:

- The requirements from the spec

- A piece of Python code from ShareLane

Also see *Black Box testing; Grey Box Testing*

**White List**

see*Dirty List - White list*

**Workaround**

Action that bypasses a problem or a way to bypass a problem

|  |  |
| --- | --- |
|  |  |