SATHYABAMA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCSX1060 – SOFTWARE TESTING (ELECTIVE) – Unit V

UNIT – DEFECT MANAGEMENT

Defects

- A defect is an error or a bug, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects.
- When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it's called differently like bug, issue, incidents or problem.
- When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect. These defects or bugs occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results.

When a tester finds a bug or defect it's required to convey the same to the developers. Thus they report bugs with the detail steps and are called as Bug Reports, issue report, problem report, etc.

This Defect report or Bug report consists of the following information:

- **Defect ID** – Every bug or defect has it's unique identification number
- **Defect Description** – This includes the abstract of the issue.
- **Product Version** – This includes the product version of the application in which the defect is found.
- **Detail Steps** – This includes the detailed steps of the issue with the screenshots attached so that developers can recreate it.
- **Date Raised** – This includes the Date when the bug is reported
- Reported By – This includes the details of the tester who reported the bug like Name and ID
- **Status** – This field includes the Status of the defect like New, Assigned, Open, Retest, Verification, Closed, Failed, Deferred, etc.
- **Fixed by** – This field includes the details of the developer who fixed it like Name and ID
- **Date Closed** – This includes the Date when the bug is closed

- **Severity** – Based on the severity (Critical, Major or Minor) it tells us about impact of the defect or bug in the software application
- **Priority –** Based on the Priority set (High/Medium/Low) the order of fixing the defect can be made. (Know more about Severity and Priority)

Defect Management

A major test objective is to identify defects. Once identified, defects need to be recorded, monitored, reported and corrected. The primary goal is to prevent defects. The defect management process like the entire software development process, should be risk driven, i.e., strategies, priorities and resources should be based on an assessment of the risk.

Defect measurement should be integrated into the development process and be used by the project team to improve the development process. Defect information should be used to improve the process Imperfect or flawed processes cause most defects.

- A defect can be defined in one of two ways

From the *producer's viewpoint*
A defect is a deviation from specifications, whether missing, wrong, or extra

From the *Customer's viewpoint*
A defect is anything that causes customer dissatisfaction, whether in the requirements or not; this is known as "fit for use."

The following structure is recommended to report a defect:
Title: *Type the problem encountered in the application, the title needs to be understandable*
For Example:
The following categories can be used.

- *Missing*
- *Inaccurate*
- *Incomplete*
- *Inconsistent*
- *Incorrect*

Example:

- Missing validation in "Project" field
- Incorrect spelling in "status" drop down list
- Description
  - *Type a brief description of the problem*
- Repro Steps:
  - *Type all the steps to get to the problem, all steps must be cleared*

For example:

1.- Login to FIDO

2.- Click on Add Invoice

3.- Type !@#$%% in Project field

4.- Click on Save

- Actual Results -*Type the actual results of the action*

For example:

The following error message is displayed..

Comments:

Type any comments or notify to the developers of any screenshots (attachments)

For Example: *This defect is reproducible in Project field*. (see attached file)

- Expected Results
  - *Type the expected results of the action.*

    For Example:

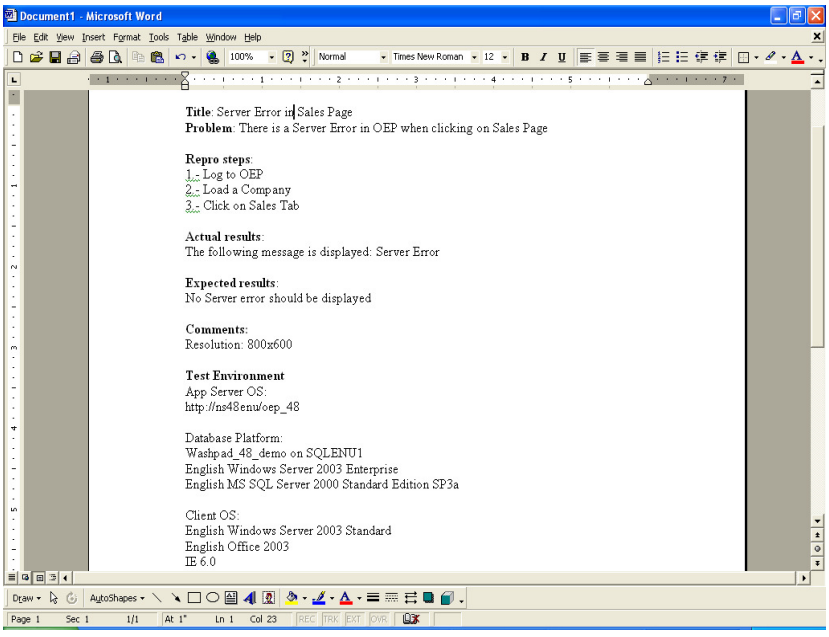
    Data should be saved successfully.
- Test Environment
  - *Include details of the test environment*

    For Example:

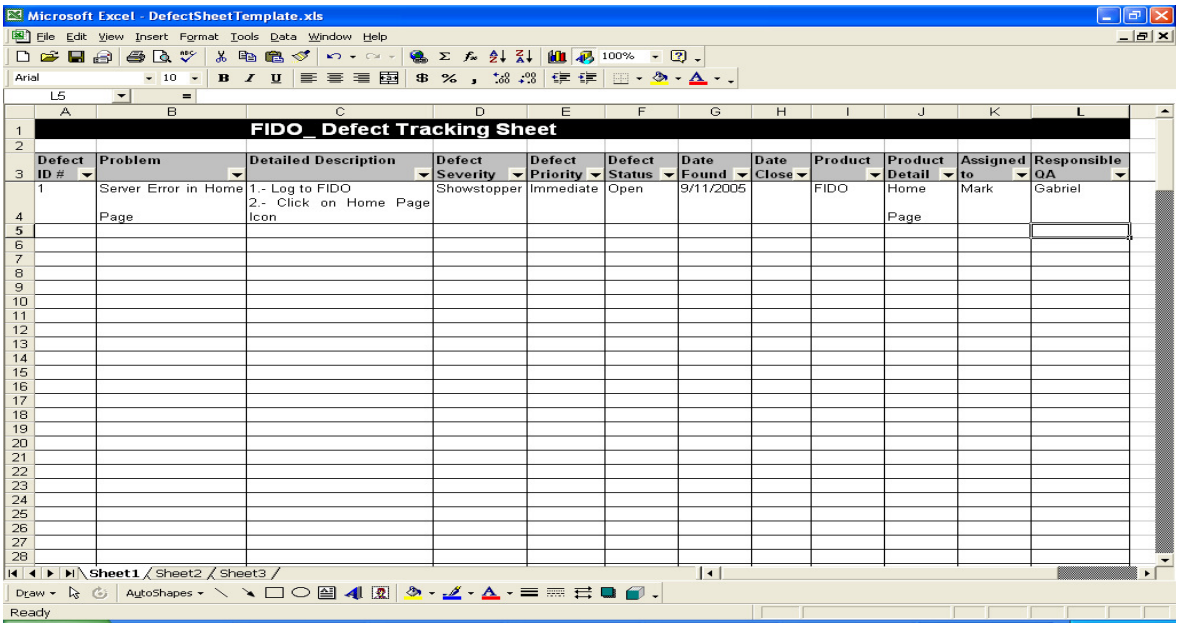
    Microsoft Windows 2003 Standard

    Office 2003

Defect report example



Defect Tracking Example



Microsoft Excel - DefectSheetTemplate.xls

File Edit View Insert Format Tools Data Window Help

Arial 10 100%

L5

FIDO_ Defect Tracking Sheet

Defect ID #	Problem	Detailed Description	Defect Severity	Defect Priority	Defect Status	Date Found	Date Close	Product	Product Detail	Assigned to	Responsible QA
1	Server Error in Home Page	1.- Log to FIDO 2.- Click on Home Page Icon	Showstopper	Immediate	Open	9/11/2005		FIDO	Home Page	Mark	Gabriel

Sheet1 Sheet2 Sheet3

Draw AutoShapes

Ready

Defect Management Process

- The steps below describe a simple defect tracking process:
  - Execute the test and compare the actual results to the documented expected results.
  - If a discrepancy exists, log the discrepancy with a status of “open”. Supplementary documentation, such as screen prints or program traces, should be attached if available.

The test manager or tester should review the problem log with the appropriate member of the development team to determine if the discrepancy is truly a defect

  - Assign the defect to a developer for correction.
  - Once the defect is corrected, the developer will usually enter a description of the fix applied and update the defect status to “Fixed” or “Retest”.
  - The defect is routed back to the test team for retesting.
  - Additional regression testing is performed as needed based on the severity and impact of the fix applied.
  - If the retest result match the expected result, the defect status is updated to “closed”. If the test results indicate that the defect is still not fixed, the status is changed to “open” and sent back to the developer.

Defect Classification and Analysis

- General Types of Defect Analyses.
- ODC: Orthogonal Defect Classification.
- Analysis of ODC Data.

Defect in Quality Data/Models

- Defect data is quality measurement data:
  - Extracted from defect tracking tools.
  - Additional (defect classification) data may be available.
  - post-release data issues
  - Defect data in quality models:
  - As results in generalized models.
  - As response/independent variables in product specific models.

General Defect Analysis

- General defect analyses: Questions
  - What? identification (and classification).
    - type, severity, etc.,
  - Where? distribution across location.
  - When? discovery/observation
    - what about when injection occurs? harder
    - pre-release: more data
    - post-release: less data, but more meaningful/sensitive
  - How/why? related to injection => use in future defect prevention.
- General defect analyses: Types
  - Distribution by type or area.
  - Trend over time.
  - Causal analysis.
  - Other analysis for classified data.

Defect Analysis: Data Treatment

- Variations of defect data:
  - Error/fault/failure perspective.
  - Pre-/post-release.
  - Unique defect?
  - Focus here: defect fixes – usually contain more data
- Why defect fixes (DF)?
  - Propagation information (system structure, component interconnection, product evolution)
  - Close ties to effort (defect fixing).
  - Pre-release: more meaningful (post release: each failure occurrence.)

Defect Distribution Analysis

- What: Distribution over ***defect*** types.
  - Types/sub-types.
  - Defect types related to product's “domain".

- Tied to quality attributes (in Ch. 2)
- Web example
- Defect = “error" in web community.
- Dominance of type E error (93%): “missing files".
- Type A error (6.76%): “permission denied” requires further analysis
- All other types: negligible

Defect distribution analysis

- Where: Distribution over *locations*.
  - Common: by *product areas*
    - Sub s product/module/procedure/etc.
    - IBM-LS: Table 20.3 (p.342) and IBM-NS: Table 20.4 (p.343)
    - common pattern: skewed distribution
  - Extension: by other *locators*
    - e.g., types of sources or code
    - example of web error distribution
    - Table 20.2 (p.342) by file type
    - again, skewed distribution!
- Distribution over other defect attributes: e.g. severity, fix type, functionality, usage scenarios, etc.
- Important observation:
  - Skewed distribution, or 80:20 rule => importance of risk identification for effective quality improvement
  - Early indicators needed! (Cannot wait after defect discoveries.)

Defect Trend Analysis

- Trend as a *continuous* function of time (or phases):
  - Similar to Putnam model (need precise time
  - Other analysis related to SRE
    - defect / effort / reliability curves (
  - Sometimes discrete analysis may be more meaningful (see examples later).
- Defect dynamics model:
  - Important variation to trend analysis.
  - Defect categorized by phase.
  - Discovery (already done).
  - Analysis to identify injection phase (difficult)
  - Focus out-of-phase/off-diagonal ones because of expense!

Defect Causal Analysis

- Defect causal analyses: Types
  - Causal relation identified:
    - error-fault vs fault-failure
    - works backwards

- Techniques: statistical or logical.

- Root cause analysis (logical):
  - Human intensive.
  - Good domain knowledge.
  - Fault-failure: individual and common.
  - Error-fault: project-wide effort focused on pervasive problems.
  - Gilb inspection has a step called process brainstorming

Statistical causal analysis: ≈ risk identification techniques

Orthogonal Defect Analysis (ODC): Overview

- Key elements of ODC
  - Aim: tracking/analysis/improve
  - Approach: classification and analysis
  - Key attributes of defects
  - Views: both failure and fault
  - Applicability: inspection and testing
  - Analysis: attribute focusing
  - Need for historical data

ODC: Why?

- Statistical defect models:
  - Quantitative and objective analyses.
  - SRGMs (Ch.22), DRM (Ch.19), etc.
  - Problems: accuracy & timeliness.
- Causal (root cause) analyses:
  - Qualitative but subjective analyses.
  - Use in defect prevention.
- **ODC solution:**
  - **Bridge the gap between the two.**
  - **Systematic scheme used.**
  - **Wide applicability.**

ODC: Ideas

- Cause-effect relation by type:
  - Different types of faults.
  - Causing different failures.
  - Need defect classification.
  - Multiple attributes for defects.
- Good measurement:
  - Orthogonality (independent view).
  - Consistency across phases.
  - Uniformity across products.
- ODC process/implementation:

- Human classification.
- Analysis method and tools.
- Feedback results (and followup).

- Classification for cause-effect or views:
  - Cause/fault: type, trigger, etc.
  - Effect/failure: severity, impact, etc.
  - Additional causal-analysis-related: source, where/when injected.

ODC Attributes: Effect/Failure-View

- Defect attribute data collected by testers at defect discovery
  - Defect trigger classes:
    - product specific
    - black box in nature – may resemble test scenario classes
    - pre/post-release triggers
  - Impact: e.g., IBM's CUPRIMDSO.
  - Severity: low-high
  - Detection time, etc.

ODC Attributes: Cause/Fault-View

- Defect attribute data collected by developers (when locating, identifying and fixing the faults)
  - Defect type
    - Associated with development process.
    - Missing or incorrect.
    - May be adapted for other products.
  - Action: add, delete, change.
  - Number of lines changed, etc.

ODC Attributes: Cause/Error-View

- Defect attribute data collected by developers (defect fixers)
  - Key attributes:
    - Defect source: vendor/base/new code.
    - Where injected.
    - When injected. (Only rough “when": phase injected.)
  - Characteristics:
    - Associated with additional causal analysis.
    - May not be performed.
    - Subjective judgment involved (evolution of ODC philosophy)

Adapting ODC

- For Web Error Analysis: Web testing/QA study.
  - Web error = observed failures, with causes already recorded in access/error logs.
  - Key attributes mapped to ODC:
    - Defect impact = web error type

        - types in Table 20.1 (p.341)
    - Defect trigger = Referring page
        - specific usage sequences or referrals
    - Defect source = specific files or file type to fix problems

May include other attributes for different kinds of applications

ODC Analysis: Attribute Focusing

- General characteristics
    - Graphical in nature
    - 1-way or 2-way distribution
    - Phases and progression
    - Historical data necessary
    - Focusing on big deviations
- Representation and analysis
    - 1-way: histograms
    - 2-way: stack-up vs multiple graphics
    - Support with analysis tools

Pareto analysis.

Pareto analysis is a formal technique useful where many possible courses of action are competing for attention. In essence, the problem-solver estimates the benefit delivered by each action, then selects a number of the most effective actions that deliver a total benefit reasonably close to the maximal possible one.[

Pareto analysis is a creative way of looking at causes of problems because it helps stimulate thinking and organize thoughts. However, it can be limited by its exclusion of possibly important problems which may be small initially, but which grow with time. It should be combined with other analytical tools such as failure mode and effects analysis and fault tree analysis for example.[

This technique helps to identify the top portion of causes that need to be addressed to resolve the majority of problems. Once the predominant causes are identified, then tools like the Ishikawa diagram or Fish-bone Analysis can be used to identify the root causes of the problems. While it is common to refer to pareto as "80/20" rule, under the assumption that, in all situations, 20% of causes determine 80% of problems, this ratio is merely a convenient rule of thumb and is not nor should it be considered immutable law of nature.

The application of the Pareto analysis in risk management allows management to focus on those risks that have the most impact on the project

Steps to identify the important causes using 80/20 rule

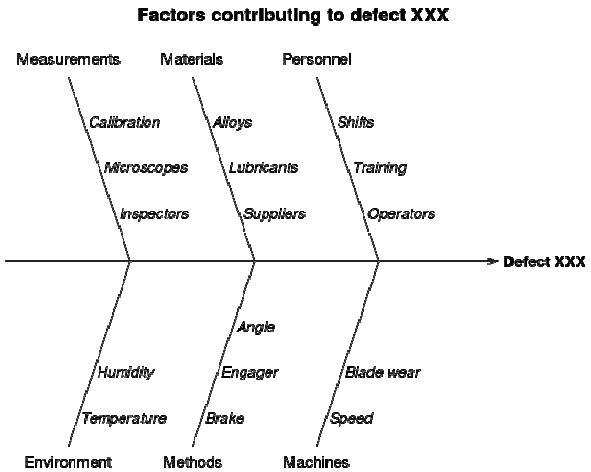
1. Form an explicit table listing the causes and their frequency of occurrence as a percentage

2. Arrange the rows in decreasing order of importance of the causes (i.e., the most important cause first)
3. Add a cumulative percentage column to the table, then plot the information
4. Plot (#1) a curve with causes on *x*- and cumulative percentage on *y*-axis
5. Plot (#2) a bar graph with causes on *x*- and percent frequency on *y*-axis
6. Draw a horizontal dotted line at 80% from the *y*-axis to intersect the curve. Then draw a vertical dotted line from the point of intersection to the *x*-axis. The vertical dotted line separates the important causes (on the left) and trivial causes (on the right)
7. Explicitly review the chart to ensure that causes for at least 80% of the problems are captured

Ishikawa diagram

Ishikawa diagrams (also called **fishbone diagrams**, **herringbone diagrams**, **cause-and-effect diagrams**, or **Fishikawa**) are causal diagrams created by Kaoru Ishikawa (1968) that show the causes of a specific event. Common uses of the Ishikawa diagram are product design and quality defect prevention to identify potential factors causing an overall effect. Each cause or reason for imperfection is a source of variation. Causes are usually grouped into major categories to identify these sources of variation. The categories typically include

- People: Anyone involved with the process
- Methods: How the process is performed and the specific requirements for doing it, such as policies, procedures, rules, regulations and laws
- Machines: Any equipment, computers, tools, etc. required to accomplish the job
- Materials: Raw materials, parts, pens, paper, etc. used to produce the final product
- Measurements: Data generated from the process that are used to evaluate its quality
- Environment: The conditions, such as location, time, temperature, and culture in which the process operates



Ishikawa diagrams were popularized in the 1960s by Kaoru Ishikawa,[3] who pioneered quality management processes in the Kawasaki shipyards, and in the process became one of the founding fathers of modern management.

The basic concept was first used in the 1920s, and is considered one of the seven basic tools of quality control.[4] It is known as a fishbone diagram because of its shape, similar to the side view of a fish skeleton.

Mazda Motors famously used an Ishikawa diagram in the development of the Miata sports car, where the required result was "Jinba Ittai" (Horse and Rider as One — jap. 人馬一体). The main causes included such aspects as "touch" and "braking" with the lesser causes including highly granular factors such as "50/50 weight distribution" and "able to rest elbow on top of driver's door". Every factor identified in the diagram was included in the final design

Causes

Causes in the diagram are often categorized, such as to the 5 M's, described below. Cause-and-effect diagrams can reveal key relationships among various variables, and the possible causes provide additional insight into process behavior.

Causes can be derived from brainstorming sessions. These groups can then be labeled as categories of the fishbone. They will typically be one of the traditional categories mentioned above but may be something unique to the application in a specific case. Causes can be traced back to root causes with the 5 Whys technique.

Typical categories are

The 5 Ms (used in manufacturing industry)

- Machine (technology)
- Method (process)
- Material (Includes Raw Material, Consumables and Information.)
- Man Power (physical work)/Mind Power (brain work): Kaizens, Suggestions
- Measurement (Inspection)

The original 5 Ms used by the Toyota Production System have been expanded by some to include the following and are referred to as the 8 Ms. However, this is not globally recognized. It has been suggested to return to the roots of the tools and to keep the teaching simple while recognizing the original intent; most programs do not address the 8Ms.

- Milieu/Mother Nature(Environment)
- Management/Money Power
- Maintenance

"Milieu" is also used as the 6th M by industries for investigations taking the environment into account.

The 8 Ps (used in marketing industry)

- Product/Service
- Price
- Place
- Promotion
- People/personnel
- Process
- Physical Evidence
- Publicity

The 8 Ps are primarily used in service marketing.

The 4 Ss (used in service industry)

- Surroundings
- Suppliers
- Systems
- Skills

Test execution engine

A **test execution engine** is a type of software used to test software, hardware or complete systems.

Synonyms of test execution engine:

- Test executive
- Test manager
- Test sequencer

A test execution engine may appear in two forms:

- Module of a test software suite (test bench) or an integrated development environment
- Stand-alone application software

Concept

The test execution engine does not carry any information about the tested product. Only the test specification and the test data carry information about the tested product.

The test specification is software. Test specification is sometimes referred to as test sequence, which consists of test steps.

The test specification should be stored in the test repository in a text format (such as source code). Test data is sometimes generated by some test data generator tool. Test data can be

stored in binary or text files. Test data should also be stored in the test repository together with the test specification.

Test specification is selected, loaded and executed by the test execution engine similarly, as application software is selected, loaded and executed by operation systems. The test execution engine should not operate on the tested object directly, but though plug-in modules similarly as an application software accesses devices through drivers which are installed on the operation system.

The difference between the concept of test execution engine and operation system is that the test execution engine monitors, presents and stores the status, results, time stamp, length and other information for every Test Step of a Test Sequence, but typically an operation system does not perform such profiling of a software execution.

Reasons for using a test execution engine:

- Test results are stored and can be viewed in a uniform way, independent of the type of the test
- Easier to keep track of the changes
- Easier to reuse components developed for testing

Functions

Main functions of a test execution engine:

- Select a test type to execute. Selection can be automatic or manual.
- Load the specification of the selected test type by opening a file from the local file system or downloading it from a Server, depending on where the test repository is stored.
- Execute the test through the use of testing tools (SW test) or instruments (HW test), while showing the progress and accepting control from the operator (for example to Abort)
- Present the outcome (such as Passed, Failed or Aborted) of test Steps and the complete Sequence to the operator
- Store the Test Results in report files

An advanced test execution engine may have additional functions, such as:

- Store the test results in a Database
- Load test result back from the Database
- Present the test results as raw data.
- Present the test results in a processed format. (Statistics)
- Authenticate the operators.

Advanced functions of the test execution engine maybe less important for software testing, but these advanced features could be essential when executing hardware/system tests.

Operations types

A test execution engine by executing a test specification, it may perform different types of operations on the product, such as:

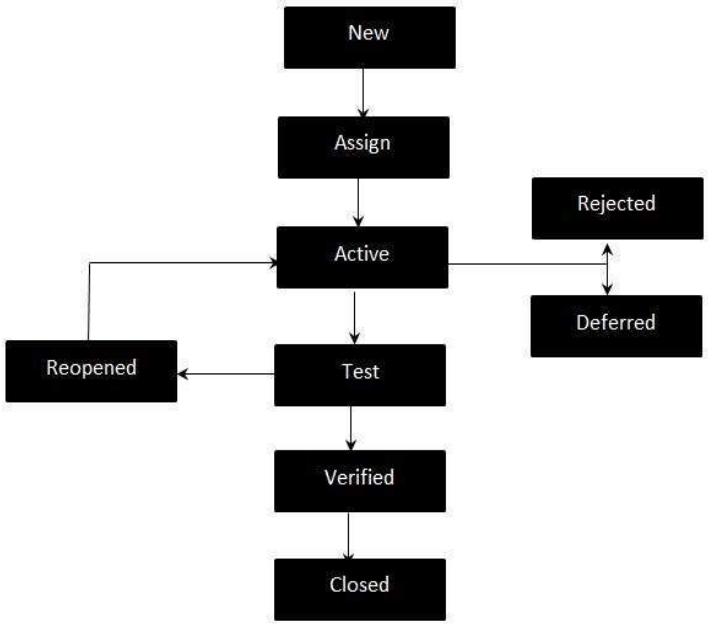
- Verification
- Calibration
- Programming
    - Downloading firmware to the product's non volatile memory (Flash)
    - Personalization: programming with unique parameters, like a serial number or a MAC address

If the subject is software, verification is the only possible operation.

What is Defect Life Cycle?

Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

Defect Life Cycle - Workflow:



Defect Life Cycle States:

- **New -** Potential defect that is raised and yet to be validated.
- **Assigned -** Assigned against a development team to address it but not yet resolved.
- **Active -** The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
- **Test -** The Defect is fixed and ready for testing.
- **Verified -** The Defect that is retested and the test has been verified by QA.

- **Closed -** The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- **Reopened -** When the defect is NOT fixed, QA reopens/reactivates the defect.
- **Deferred -** When a defect cannot be addressed in that particular cycle it is deferred to future release.
- **Rejected -** A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

Bugzilla

Bugzilla is an open-source issue/bug tracking system that allows developers effectively to keep track of outstanding problems with their product. It is written in Perl and uses MYSQL database. Bugzilla is a defect tracking tool, however it can be used as a test management tool as such it can be easily linked with other test case management tools like Quality Center, Testlink etc.

This open bug-tracker enables users to stay connected with their clients or employees, to communicate about problems effectively throughout the data-management chain.

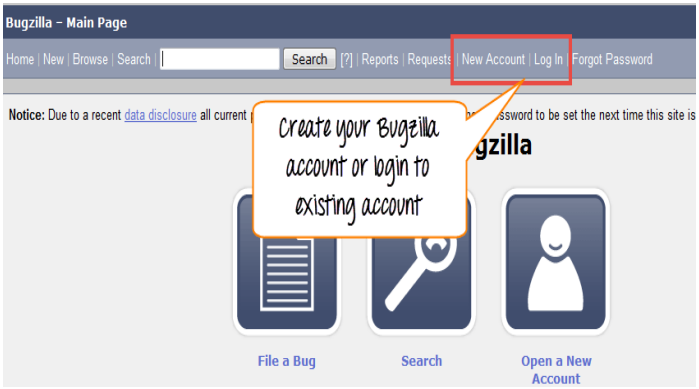
Bugzilla – Developed by Mozilla, this open source testing tool works as the name suggests. Bugzilla specializes in detecting bugs found in the application or website. Since the application is open-source it can be used freely and it is availability in different OS makes it even a viable alternative for error tracking. The only downside is it has a long list of requirements before it could run.

Key features of Bugzilla includes

- Advanced search capabilities
- E-mail Notifications
- Modify/file Bugs by e-mail
- Time tracking
- Strong security
- Customization
- Localization

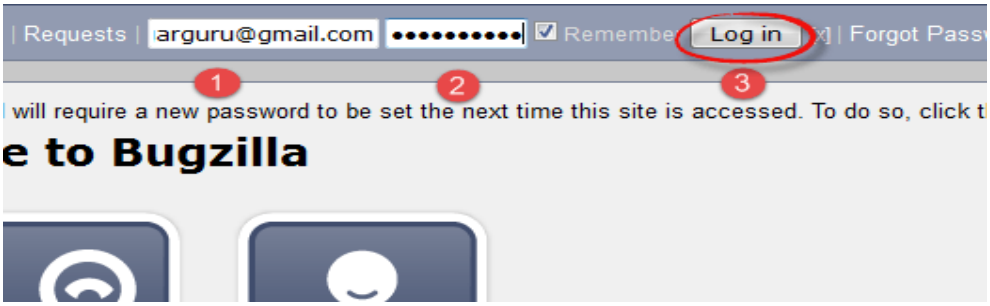
How to log-in to Bugzilla

Step 1) To create an account in Bugzilla or to login into the existing account go to **New Account or Log in** option in the main menu.

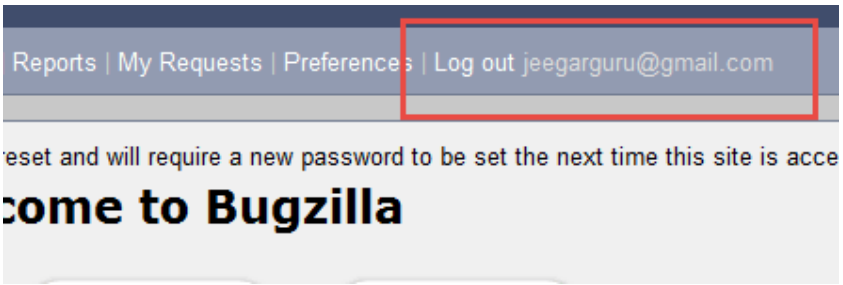


Step 2) Now, enter your personal details to log into Bugzilla

1. User ID
2. Password
3. And then click on **"Log in"**



Step 3) You are successfully logged into Bugzilla system



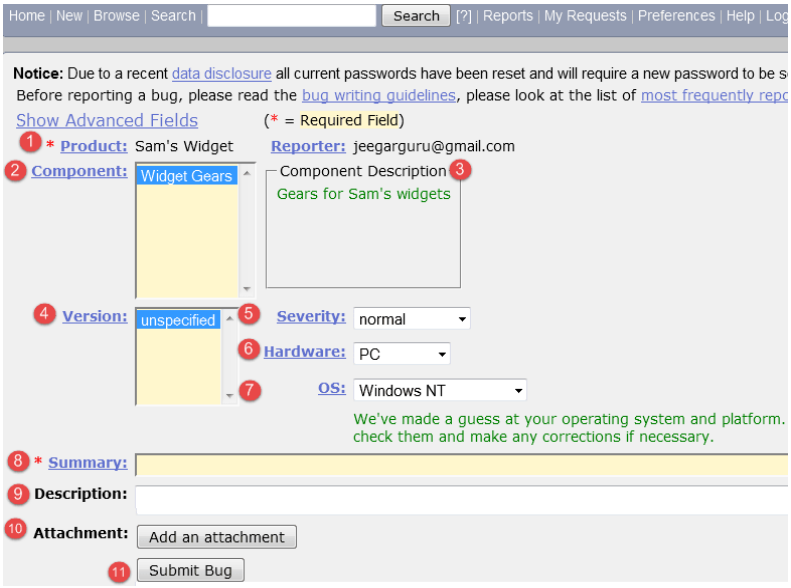
Creating a Bug-report in Bugzilla

Step 1) To create a new bug in Bugzilla, visit the home-page of Bugzilla and click on **NEW** tab from the main menu

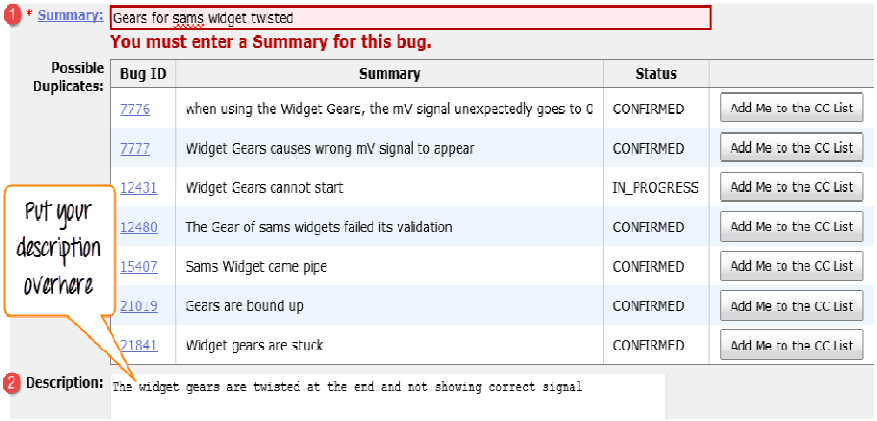
Step 2) In the next window

1. Enter Product
2. Enter Component
3. Give Component description
4. Select version,
5. Select severity
6. Select Hardware
7. Select OS
8. Enter Summary
9. Enter Description
10. Attach Attachment
11. Submit

NOTE: The above fields will vary as per your customization of Bugzilla

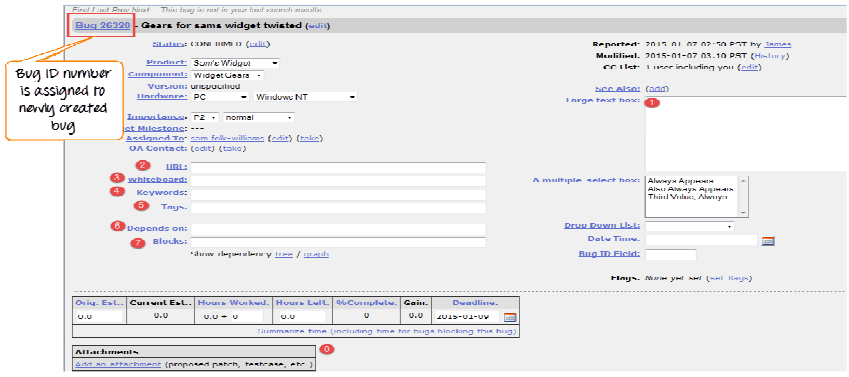


NOTE: The mandatory fields are marked with *. Example : Summary, Description

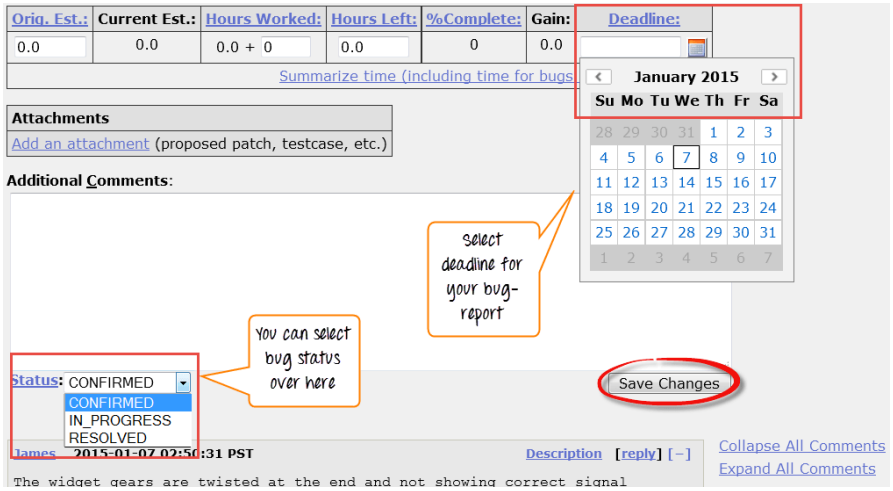


Step 4) Bug is created ID# 26320 is assigned to our Bug. You can also add additional information to the assigned bug like URL, keywords, whiteboard, tags, etc. This extra-information is helpful to give more detail about the Bug you have created.

1. Large text box
2. URL
3. Whiteboard
4. Keywords
5. Tags
6. Depends on
7. Blocks
8. Attachments



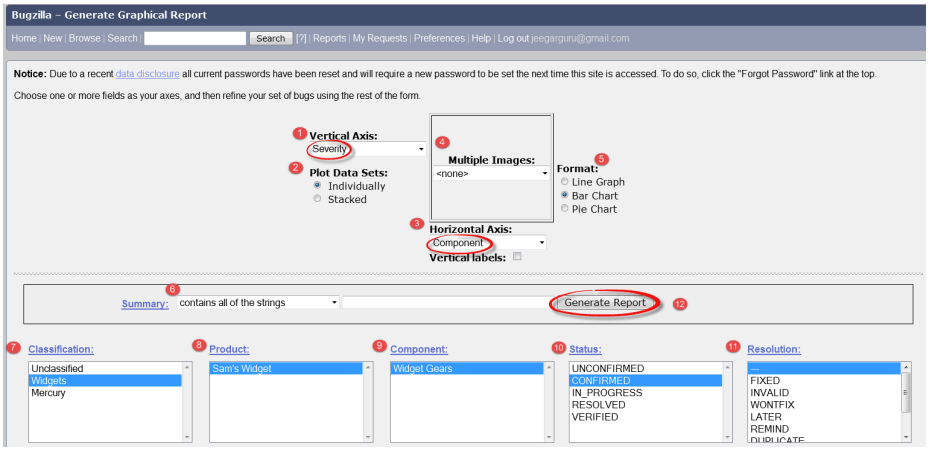
Step 5) In the same window if you scroll down further. You can select deadline date and also status of the bug. **Deadline in Bugzilla usually gives the time-limit to resolve the bug in given time frame.**



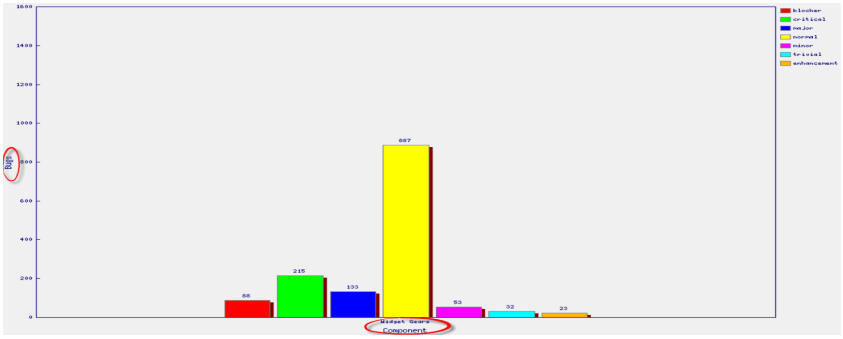
Create Graphical Reports

Graphical reports are one way to view the current state of the bug database. You can run reports either through an HTML table or graphical line/pie/bar-chart-based one. The idea behind graphical report in Bugzilla is to define a set of bugs using the standard search interface and then choosing some aspect of that set to plot on the horizontal and vertical axes. You can also get a 3-dimensional report by choosing the option of "Multiple Pages".

Reports are helpful in many ways, for instance if you want to know which component has the largest number of bad bugs reported against it. In order to represent that in the graph, you can select severity on X-axis and component on Y-axis, and then click on generate report. It will generate a report with crucial information.



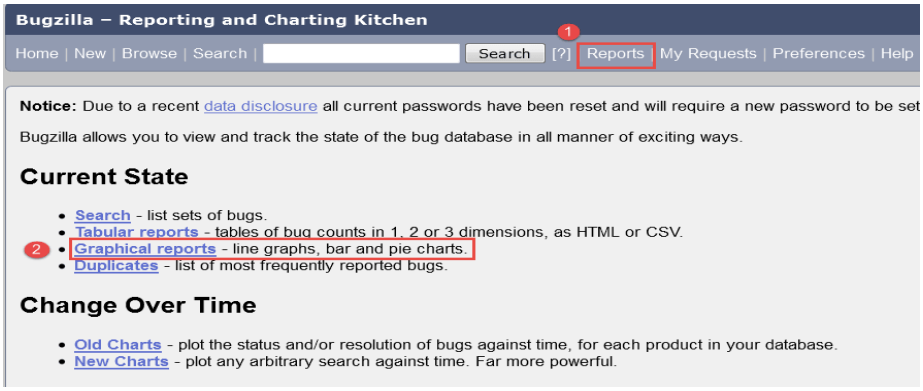
The graph below shows the Bar chart representation for the Bugs severity in component **"Widget Gears".** In the graph below, the most severe bug or blockers in components are 88 while bugs with normal severity is at top with 667 number.



Likewise, we will also see the line graph for **%complete Vs Deadline**

Step 1) To view your report in a graphical presentation,

- Click on Report from Main Menu
- Click on the Graphical reports from the given option



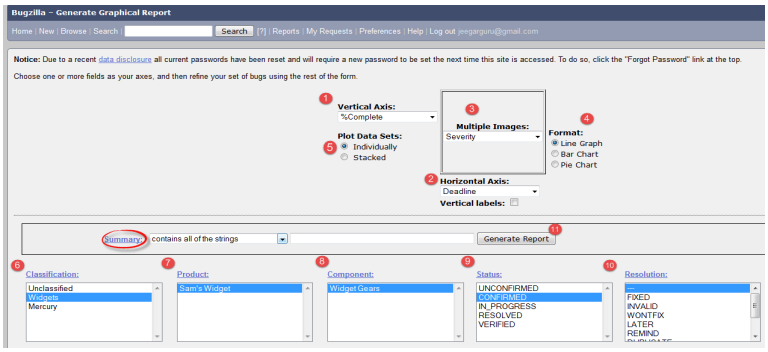
Step 2) Let's create a graph of **% Complete Vs Deadline**

In here on the vertical axis we chose **% Complete** and on our horizontal axis we chose **Deadline**. This will give the graph of amount of work done in percentage against the set-deadline.

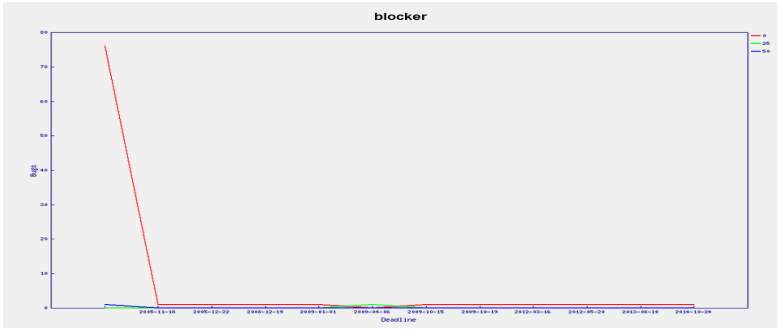
Now, set various option to present reports graphically

1. Vertical Axis
2. Horizontal Axis
3. Multiple Images
4. Format- Line graph, Bar chart or Pie chart
5. Plot data set
6. Classify your bug
7. Classify your product
8. Classify your component
9. Classify bug status
10. Select resolution

11. Click on generate report



The image of the graph will appear somewhat like this



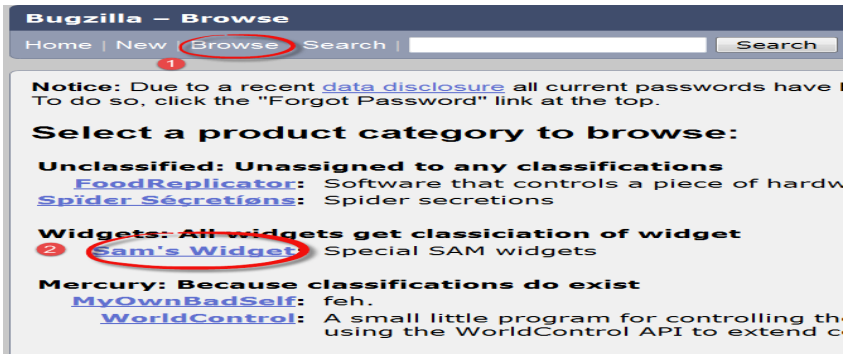
Browse Function

Step 1) To locate your bug we use browse function, click on **Browse** button from the main menu.

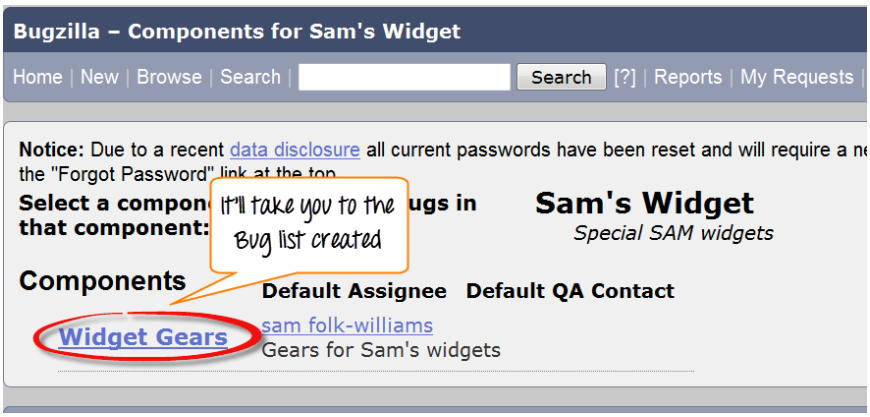
Step 2) As soon as you click on browse button a window will open saying **"Select a product category to browse"** as shown below, we browse the bug according to the category.

- After clicking the browse button

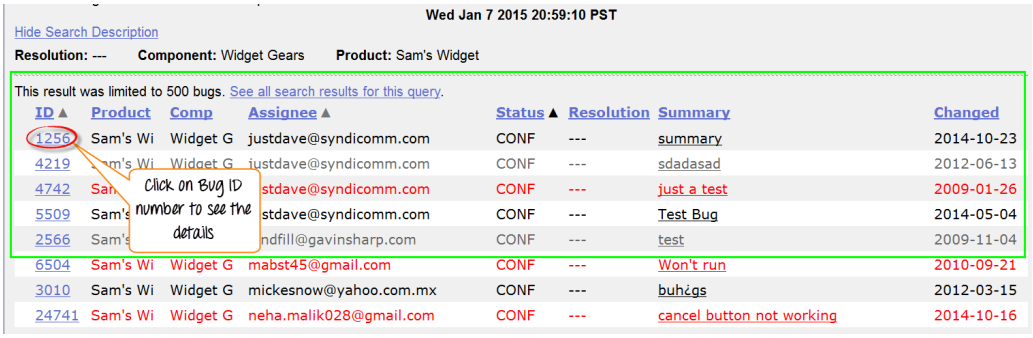
- Select the product "Sam's Widget" as such you have created a bug inside it



Step 3) It opens another window, in this click on component **"widget gears"**. Bugzilla Components are sub-sections of a product. For instance, here our product is **SAM'S WIDGET** whose component is **WIDGET GEARS**.



Step 4) when you click on the component, it will open another window. All the Bugs created under particular category will be listed over-here. From that Bug-list, choose your Bug#ID to see more details about the bug.



It will open another window, where information about your bug can be seen more in detail. In the same window, you can also change the assignee, QA contact or CC list.


Bug 1256 - summary (edit)
Save Changes
Status: CONFIRMED (edit)
Product: Sam's Widget
Component: Widget Gears
Version: unspecified
Hardware: HP Linux
Importance: P1 normal
Target Milestone: ---
Assigned To: Dave Miller (edit) (take)
QA Contact: (edit) (take)
URL:
Whiteboard:
You can add users to CC list from here
You can change the assignee or QA contact from here itself
Reported: 2003-05-27 13:28 PDT by Jason McCallum
Modified: 2014-10-23 08:09 PDT (History)
CC List: Add me to CC list
6 users (edit)
See Also: (add)
Large text box:
free text:
A multiple-select box:
Always Appears
Also Always Appears
Third Value, Always