process does not end up extracting a large subset of data that cannot be imported into the test environment, due to size limitations.

The benefits of having a test data management is that it can:

- Keep data management costs low with smaller sets of data that require less storage and fewer computing resources.
- Assure confidentiality of sensitive data.
- Assure privacy of information by not importing or masking private information.
- Reduce the likelihood of insider threats and frauds.

Defect Reporting and Tracking

Coding bugs, design flaws, behavioral anomalies (logic flaws), errors, faults and vulnerabilities all constitute software defects as depicted in *Figure 5.13* and once any defect is suspected and/or identified, it needs to be appropriately reported, tracked and addressed, prior to release. In this section, we will focus on how to report and track software defects. In the following section, we will learn about how these defects can be addressed based upon the potential impact they have and what corrective actions can be taken.

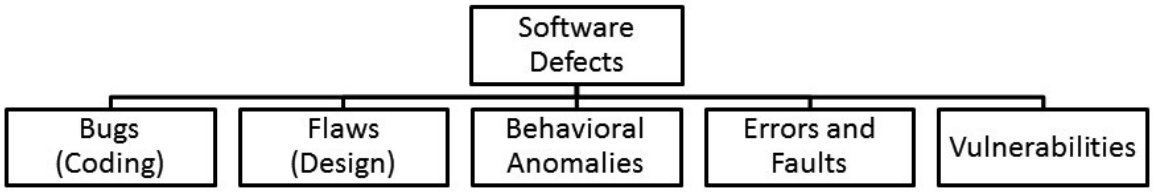

Figure 5.13 – **Software Defects**

Software defects need to be first reported and then tracked. Reporting defects must be comprehensive and detailed enough to provide the software development teams the information that is necessary to determine the root cause of the issue, so that they can address it.

Reporting Defects

The goal of reporting defects is to ensure that they get addressed. Information that must be included in a defect report is:

Defect Identifier (ID)

A unique number or identifier must be given to each defect report so that each defect can be tracked appropriately. Don't try to clump multiple issues into one

defect. Each issue should warrant its own defect report. Most defect tracking tools have an automated means to assign a defect ID when a new defect is reported.

Title

Provide a concise yet descriptive title for the defect. For example, 'Image upload fails'

Description

Provide a summary of the defect to elaborate on the defect title you specified. For example, you can say, 'When attempting to insert an image into a blog, the software does not allow the upload of the image and fails with an error message'.

Detailed Steps

If the defect is not reproducible then the defect will not get fixed. This is the reason why detailed steps as to how the defect can be reproduced by the software development team is necessary. For example, it is not sufficient to say that the 'Upload' feature does not work. Instead, it is important to list out the steps taken by the tester, such as:

- Provided username and password and clicked on 'Log in'.
- Upon successful authentication, clicked on 'New blog'.
- Entered blog title as 'A picture is worth a thousand words' in 'Title' field.
- Entered description as 'Please comment on the picture you see' in the 'Description' field.
- Clicked on the 'Upload image' icon.
- Clicked on the 'Browse' button in the 'Image Upload' pop up screen.
- Browsed to the directory and selected the image to upload and clicked 'Open' in the 'Browse Directory' pop up window.
- The 'Browse Directory' windows closed and the 'Image Upload' pop up screen got focus.
- Clicked on the button 'Upload' in the 'Image Upload' pop up screen.
- An error message was shown stating that the upload directory could not be created and the Upload failed.

Expected Results

It is important to describe what the expected result of the operation is so that the development teams can understand the discrepancy from intended functionality. The best way to do this is to tie the defect ID with the requirement identifier in the Requirements Traceability Matrix (RTM). This way any deviations from intended functionality as specified in the requirements can be reviewed and verified against.

Screenshot

If possible and available, a screenshot of the error message should be attached. This proves very helpful to the software development team for the following reasons:

- It provides the development team members a means to visualize the defect symptoms that the tester reports.
- It assures the development team members that they have successfully reproduced the same defect that the tester reported.

An example of a screenshot is depicted in *Figure 5.14*. ***Note -*** if the screenshot image contains sensitive information, it is advisable to not capture the screenshot in the first place. If however, a screenshot is necessary, then appropriate security controls such as masking of the sensitive information in the defect screenshot or role based access control should be implemented to protect against disclosure threats.

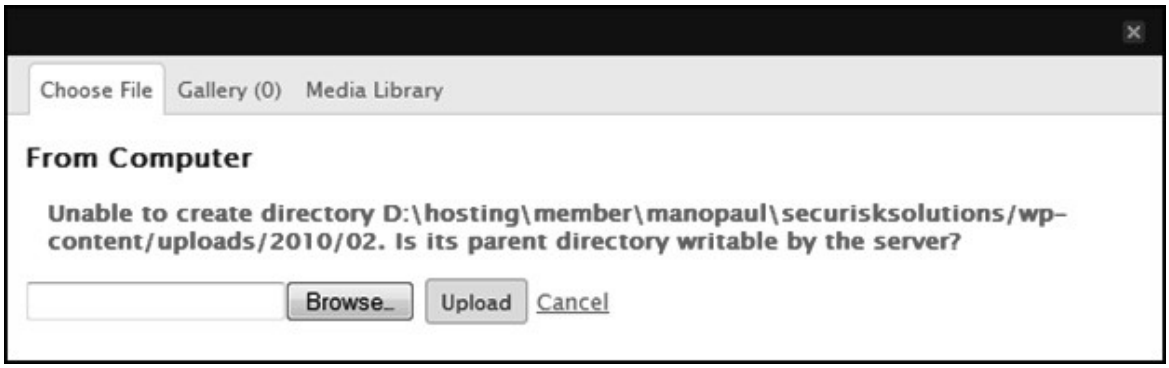

Figure 5.14 - **Defect Screenshot**

Type

If possible, it is recommended to categorize the defect based on whether it is a functional issue or an assurance (security) one. You can also sub-categorize the defect. *Figure 5.15* is an example of categories and sub-categories of software defects.

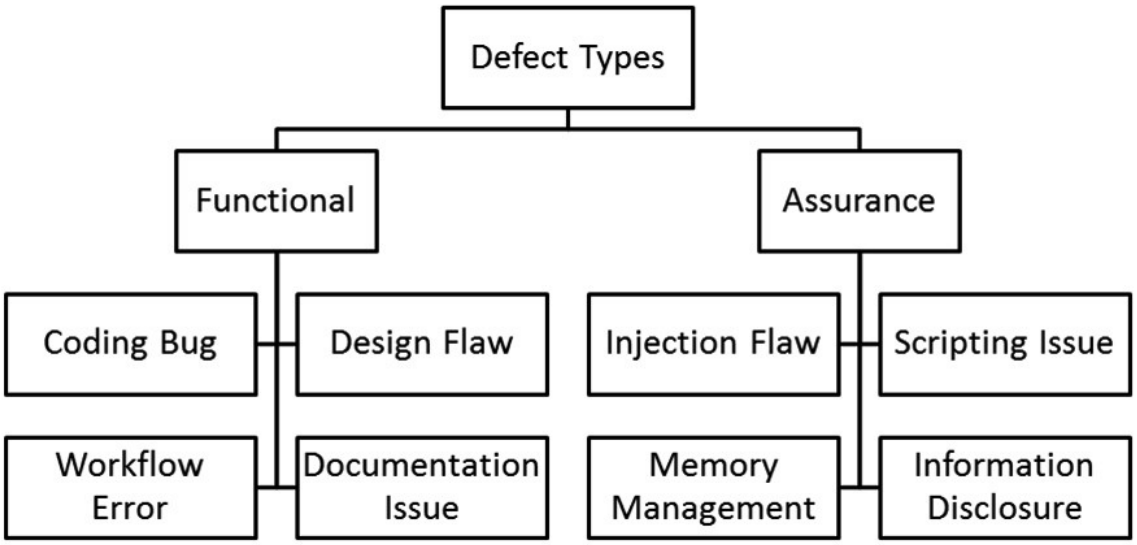

Figure 5.15 – **Defect Types**

This way, pulling reports on the types of defects in the software is made easy. Furthermore, it makes it easy to find out the security defects that are to be addressed prior to release.

Environment

Capturing the environment in which the defect was evident is important. Some important considerations to report on include:

- Was it in the test environment or was it in the production environment?
- Was the issue evident only in one environment?
- Was the issue determined in the intranet, extranet or Internet environment?
- What is the Operating System and the service pack on which the issue was experienced? Are systems with other service packs experiencing the same issue?
- Was this a web application issue and if so, what was the web address?

Build Number

The version of the product in which the defect was determined is an important aspect in defect reporting. This makes it possible to compare versions and see if the defect is universal or specific to a particular version. From a security perspective, the build number can be used to determine the RASQ between versions, based on the number of security defects that are prevalent in each version release.

Tester Name

The individual who detected the defect must be specified so that the development team members know whom they need to contact for clarification or further information.

Reported On

The date and time (if possible) as to when the defect was reported needs to be specified. This is important in order to track the defect throughout its life cycle (covered later in this chapter) and determine the time it takes to resolve a defect, as a means to identify process improvement opportunities.

Severity

This is to indicate the tester's determination of the impact of the defect. This may or may not necessarily be the actual impact of the defect, however it provides the remediation team with additional information that is necessary to prioritize their efforts. This is often qualitative in nature and some examples of severity types are:

- ***Critical*** – the impact of the defect will not allow the software to be functional as expected. All users will be affected.
- ***Major*** – Some of the expected business functionality has been affected and operations cannot continue, since there is no work-around available.
- ***Minor*** – Some of the expected business functionality has been affected but operations can continue because a work-around is in place.
- ***Trivial*** – Business functionality is not affected but can be enhanced with some changes that would be nice to have. UI enhancements usually fall into this category.

Priority

The priority indicator is directly related to the extent of impact (severity) of the defect and is assigned based on the amount of time within which the defect needs to be addressed. It is a measure of urgency and supports the availability tenet of software assurance. Some common examples of priority include, Mission Critical (0-4 hours), High (>4-24 hours), Medium (>24-48 hours) and Low (>48 hours).

Status

Every defect that is reported automatically starts with the 'New' status and as it goes through its life cycle the status is changed from 'New' to 'Confirmed',

'Assigned', 'Work-in-progress', 'Resolved/Fixed', 'Fix verified', 'Closed', 'Reopened', 'Deferred', etc.

Assigned to

When a software defect is assigned to a development team member so that is can be fixed, the name of the individual who is working the issue must be specified.

Tracking Defects

Upon the identification and verification of a defect, the defect needs to be tracked so that it can be addressed accordingly. It is advisable to track all defects related to the software in a centralized repository or defect tracking system. Centralization of defects makes it possible to have a comprehensive view of the software functionality and security risk. It also makes it possible to ensure that no two individuals are working on the same defect. A defect tracking system should have the ability to support the following requirements:

- *Defect documentation* – All required fields from a defect report must be recorded. In situations where additional information needs to be recorded, the defect tracking system must allow for the definition of custom fields.
- *Integration with Authentication Infrastructure* – A defect tracking system that has the ability to automatically fill the authenticated user information by integrating with the authentication infrastructure is preferred to prevent user entry errors. It also makes it possible to track user activity as they work on a defect.
- *Customizable Workflow* – A software defect continues to be a defect until it has been fixed or addressed. Each defect goes through a life cycle, an example of which is depicted in *Figure 5.16*. As the software defect moves from one status to another, workflow information pertinent to that defect must be tracked and, if needed, customized.
- *Notification* – When a software defect state moves from one status to another, it would be necessary to notify the appropriate personnel of the change so that processes in the SDLC are not delayed. Most defect tracking systems provide a notification interface that is configurable with whom and what to notify upon status change.
- *Auditing capability* – For accountability reasons, all user actions within the software defect tracking system must be audited and the software defect tracking system must allow for storing and reporting on these auditable information in a secure manner.

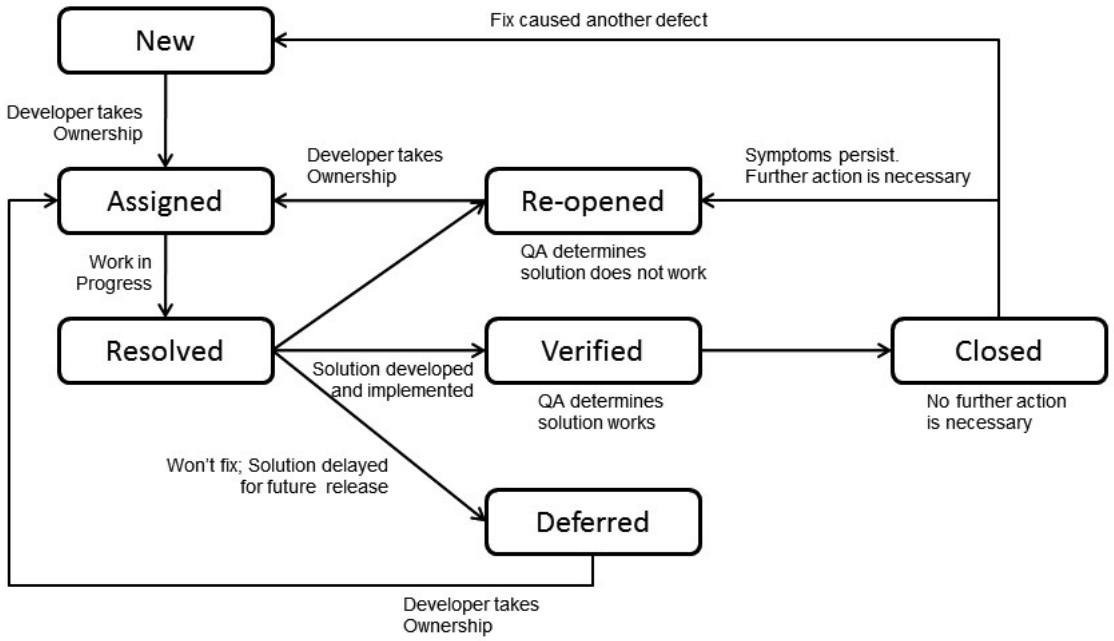

Figure 5.16 - **Defect Life Cycle**

Impact Assessment and Corrective Action

Testing findings that are reported as defects needs to be addressed. We can use the priority (urgency) and severity (impact) levels from the defect report to address software defects. High impact, high risk defects are to be addressed first. When agile or extreme programming methodologies are used, identified software defects need to be added to the backlog. Risk management principles (covered in the Secure Software Concepts chapter) can be used to determine how the defect is going to be handled. Corrective actions have a direct bearing on the risk. These can include one or more of the following:

- Fixing the defect (mitigating the risk),
- Deferring the functionality (not the fix) to a latter version (transferring the risk)
- Replacing the software (avoiding the risk)

Knowledge of security defects in the software and ignoring the risk can have serious and detrimental effects when the software is breached. All security defects must be addressed and preferably mitigated.

Additionally, it is important to fix the defects in the development environment, attest the solution in the testing environment, and verify functionality in the UAT environment and only then release (promote) the fix to production environments as illustrated in *Figure 5.17*.

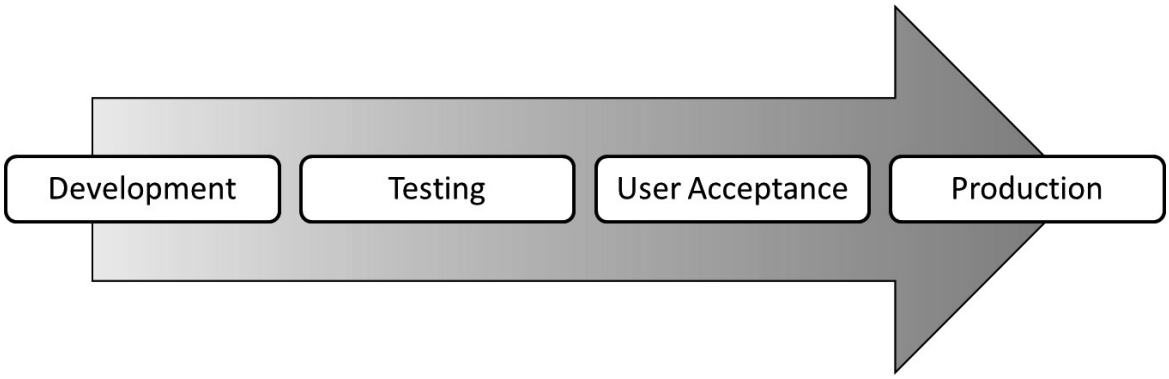

Figure 5.17 – **Fixing defects environment and process**

The following references are recommended to get additional information on secure software testing concepts and techniques:

- (ISC)² whitepaper entitled "Assuring Software Security Through Testing: White, Black and somewhere in between." provides some excellent guidance on attesting software assurance, and covers the different types of testing as it pertains to security and functionality.
- SP 800-92 published by NIST provides guidance on log management and insight into how to protect audit trails and ensuring the management of security logs.
- The MSDN article on 'Generating Test Data for Database by Using Data Generators' gives insight into leveraging the existing Integrated Development Environments to generate test data as a means to manage test data.