
Testing Concepts
Lesson 2: Testing throughout the Software Development Life Cycle
Capgemini

Lesson Objectives

To understand the following topics :

- Software Development Lifecycle Models
  - Software Development and Software Testing
  - Software Development Lifecycle Models in Context
- Test Levels
  - Component Testing
  - Integration Testing
  - System Testing
  - Acceptance Testing

Lesson Objectives

- Test Types
  - Functional Testing
  - Non-functional Testing
  - White-box Testing
  - Change-related Testing
  - Test Types and Test Levels
- Maintenance Testing
  - Triggers for Maintenance
  - Impact Analysis for Maintenance
- Test Case Terminologies
- Test Data

2.1 Software Development Life Cycle (SDLC) Models

Testing is not a stand-alone activity

It has its place within a SDLC model

In any SDLC model, a part of testing is focused on Verification and a part is focused on Validation

- Verification: Is the deliverable built according to the specification?
- Validation: Is the deliverable fit for purpose?

A SDLC model describes the types of activity performed at each stage in a software development project, and how the activities relate to one another logically and chronologically.
There are a number of different software development lifecycle models, each of which requires different approaches to testing.

- Waterfall model
- Rapid Application Development Model (RAD)
- RUP (Rational Unified Process)
- V model
- Agile Model

2.1.1 Software Development & Software Testing

In any SDLC model, there are several characteristics of good testing:

- For every development activity, there is a corresponding test activity
- Each test level has test objectives specific to that level
- Test analysis and design for a given test level begin during the corresponding development activity
- Testers participate in discussions to define and refine requirements and design, and are involved in reviewing work products (e.g., requirements, design, user stories, etc.) as soon as drafts are available

No matter which SDLC model is chosen, test activities should start in the early stages of the lifecycle, adhering to the testing principle of early testing.

2.1.2 Software Development Lifecycle Models in Context

- SDLC models must be selected and adapted based on the context of project and product characteristics such as - project goal, the type of product being developed, business priorities (e.g., time-tomarket), and identified product and project risks.

  Example : The development and testing of a minor internal administrative system should differ from the development and testing of a safety-critical system such as an automobile's brake control system.

  Example : In some cases organizational and cultural issues may inhibit communication between team members, which can impede iterative development.

2.1.2 Software Development Lifecycle Models in Context (Cont.)

- Depending on the context of the project, it may be necessary to combine or reorganize test levels and/or test activities.

  Example : For the integration of a commercial off-the-shelf (COTS) software product into a larger system, the purchaser may perform interoperability testing at the system integration test level (e.g., integration to the infrastructure and other systems) and at the acceptance test level (functional and non-functional, along with user acceptance testing and operational acceptance testing).

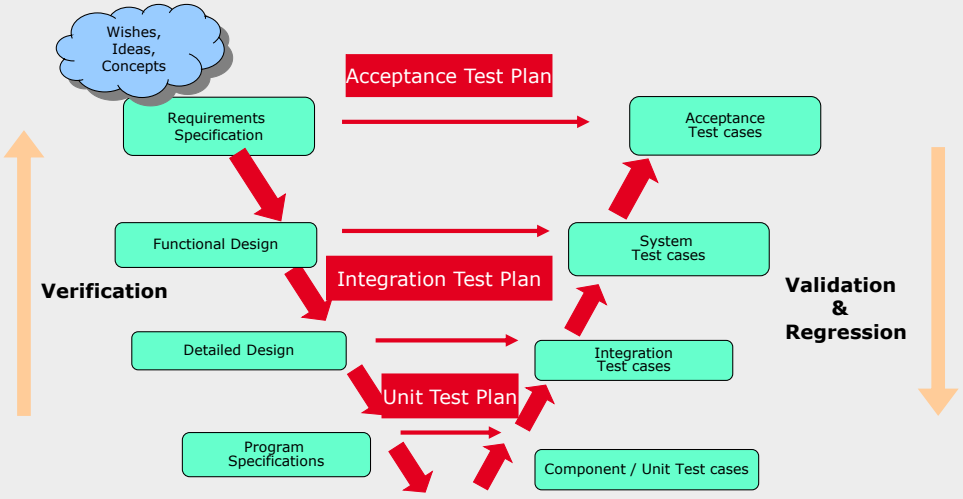
In addition, SDLC models themselves may be combined.

Example : V model may be used for the development and testing of the backend systems and their integrations, while an Agile development model may be used to develop and test the front-end user interface (UI) and functionality. Prototyping may be used early in a project, with an incremental development model adopted once the experimental phase is complete.

Internet of Things (IoT) systems, which consist of many different objects, such as devices, products, and services, typically apply separate SDLC models for each object. This presents a particular challenge for the development of Internet of Things system versions. Additionally the SDLC of such objects places stronger emphasis on the later phases of the SDLC after they have been introduced to operational use (e.g., operate, update, and decommission phases).

2.2 Test Levels in V-Model

- Test levels are groups of test activities that are organized and managed together. Each test level is an instance of the test process related to other activities with in SDLC.



V Model :

- The V proceeds from left to right, depicts the basic sequence of development and testing activities
- Left side shows development activities.
- Right side shows Testing activities
- Specific Testing activities are carried in parallel to development activities.
- The V model is valuable because it highlights the existence of several levels or phases of testing and depicts the way each relates to a different development phase.

Verification and Validation

Verification

- Verification refers to a set of activities which ensures that software correctly implements a specific function.
- Purpose of verification is to check: Are we building the product right?
- Example: code and document reviews, inspections, walkthroughs.
- It is a Quality improvement process.
- It is involve with the reviewing and evaluating the process.
- It is conducted by QA team.
- Verification is Correctness.

V&V encompasses many of the activities that are encompassed by S/w quality assurance that include formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, installation testing.

Example of Verification : code and document inspections, walkthroughs, and other techniques. unit testing , integration testing , system testing

If we are in a shopping centre and buy a thing with a code number 2342 and when we go to till and they check the number of that item and find it wrong then system will check all product number of the relevant number but don't find any number of this kind then we can say that the verify thing is wrong.

Verification is a process, which performs testing to ensure implemented functions meeting to designed functions.

Verification and Validation (Cont.)

Validation

- Validation is the following process of verification.
- Purpose of Validation is to check : Are we building the right product?
- Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
- After each validation test has been conducted, one of two possible conditions exist:
  1. The function or performance characteristics conform to specification and are accepted, or
  2. Deviation from specification and a deficiency list is created.
- It is conducted by development team with the help from QC team.

Validation ensures the correct functionality.
Validation of software typically includes evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements.

2.2 Test Levels (Cont.)

Unit (Component) testing

- Unit testing is code-based and performed primarily by developers to demonstrate that their smallest pieces of executable code function suitably.

Integration testing

- Integration testing demonstrates that two or more units or other integrations work together properly, and tends to focus on the interfaces specified in low-level design.

System testing

- System testing demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design.

Acceptance testing

- Acceptance testing is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.

2.2 Test Levels (Cont.)

Test levels are characterized by the following attributes:

- Specific objectives
- Test basis, referenced to derive test cases
- Test object (i.e., what is being tested)
- Typical defects and failures
- Specific approaches and responsibilities.

For every test level, a suitable test environment is required.

Example : In acceptance testing, for example, a production-like test environment is ideal, while in component testing the developers typically use their own development environment.

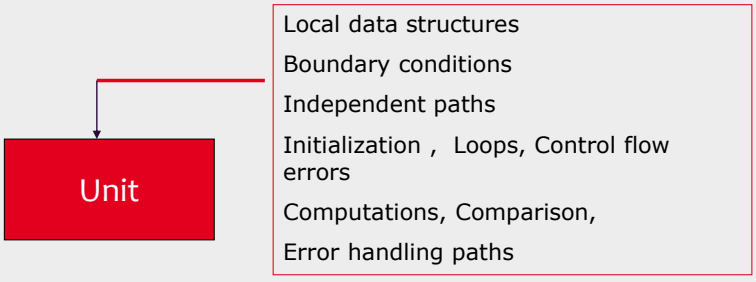
2.2.1 Component Testing

- The most 'micro' scale of testing to test particular functions, procedures or code modules or components is called Component testing ; Also called as Module or Unit testing
- Typically done by the programmer and not by Test Engineers, as it requires detailed knowledge of the internal program design and code.
- Objectives of component testing include:
  - Reducing risk
  - Verifying whether the functional and non-functional behaviors of the component are as designed and specified
  - Building confidence in the component's quality
  - Finding defects in the component
  - Preventing defects from escaping to higher test levels

In some cases, especially in incremental and iterative development models (e.g., Agile) where code changes are ongoing, automated component regression tests play a key role in building confidence that changes have not broken existing components.
Component testing is often done in isolation from the rest of the system, depending on the SDLC model and the system, which may require mock objects, service virtualization, harnesses, stubs, and drivers.
Component testing may cover functional (e.g., correctness of calculations), non-functional characteristics (e.g., searching for memory leaks), and structural properties (e.g., decision testing).

Component /Unit Testing

Unit testing uncovers errors in logic and function within the boundaries of a component.



The module interface is tested to ensure that information properly flows into and out of the program under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.

Test cases should uncover errors such as

(1) comparison of different data types
(2) incorrect logical operators or precedence
(3) expectation of equality when precision error makes equality unlikely
(4) incorrect comparison of variables
(5) improper or nonexistent loop termination
(6) failure to exit when divergent iteration is encountered.

Component Testing

Test Basis : Test basis for component testing include:

- Detailed design
- Code
- Data model
- Component specifications

Test objects : Typical test objects for component testing include:

- Components, units or modules
- Code and data structures
- Classes
- Database modules

Typical defects and failures : Typical defects and failures for component testing include :

- Incorrect functionality (e.g., not as described in design specifications)
- Data flow problems
- Incorrect code and logic

Specific approaches and responsibilities in Component Testing :
Component testing is usually performed by the developer who wrote the code. Developers will often write and execute tests after having written the code for a component. However, in Agile development especially, writing automated component test cases may precede writing application code.

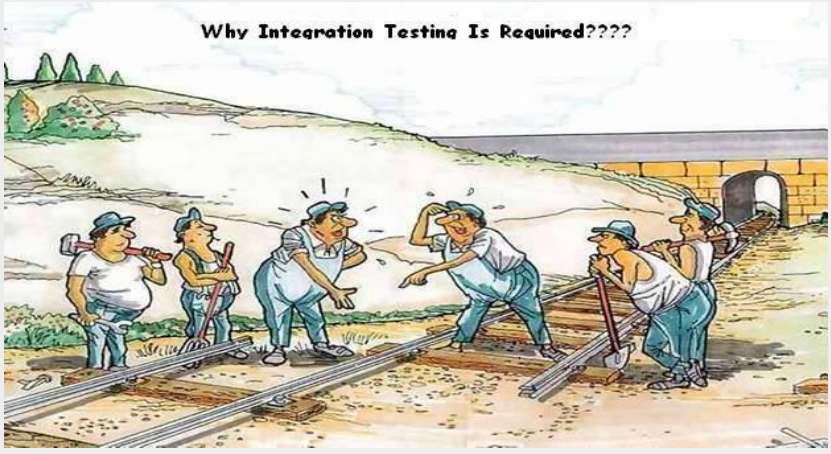
For example, consider test driven development (TDD). It is highly iterative and is based on cycles of developing automated test cases, then building and integrating small pieces of code, then executing the component tests, correcting any issues, and re-factoring the code. This process continues until the component has been completely built and all component tests are passing.
TDD is an example of a test-first approach. While TDD is originated in eXtreme Programming (XP), and it has spread to other forms of Agile.

2.2.2 Integration testing

- Integration testing focuses on interactions between components or systems.
- Objectives of Integration testing include:
  - Reducing risk
  - Verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified
  - Building confidence in the quality of the interfaces.
  - Finding defects (which may be in the interfaces themselves or within the components or systems)
  - Preventing defects from escaping to higher test levels

Interfaces are the means by which data is passed to and from modules. "Interface integrity" – to ensure that when data is passed to another module, by way of a call, none of the data becomes lost or corrupted. This loss or corruption can happen by number of ways- calling and receiving parameters may be of the wrong type and so the data appears in the receiving programs in a garbled form; there may be different number of calling and receiving parameters and so the data is lost; arrays may be of different lengths.
Global data structures are those pieces of data, maybe in the form of files, databases or variables, that are existing through out the entire system. Every module has access to global data structures and may alter the contents thereof. The effect of this again may create some unusual combination of data which reveals an error in another module.

Why Integration Testing is Required?



Two Levels of Integration testing

There are two different levels of integration testing which may be carried out on test objects of varying size as follows:

- **Component integration testing (CIT)** focuses on the interactions and interfaces between integrated components. Component integration testing is performed after component testing.
- **System integration testing (SIT)** focuses on the interactions and interfaces between systems, packages, and micro services. System integration testing may be done after system testing.

Component integration testing focuses on the interactions and interfaces between integrated components. Component integration testing is performed after component testing, and is generally automated. In iterative and incremental development, component integration tests are usually part of the continuous integration process. **System integration testing** focuses on the interactions and interfaces between systems, packages, and microservices. System integration testing can also cover interactions with, and interfaces provided by, external organizations (e.g., web services). In this case, the developing organization does not control the external interfaces, which can create various challenges for testing (e.g., ensuring that test-blocking defects in the external organization's code are resolved, arranging for test environments, etc.). System integration testing may be done after system testing or in parallel with ongoing system test activities (in both sequential development and iterative and incremental development).

Integration Testing

Test Basis : Test basis for Integration testing include:

- Software and system design
- Sequence diagrams
- Interface and communication protocol specifications
- Use cases
- Architecture at component or system level
- Workflows
- External interface definitions

Test objects : Typical test objects for Integration testing include:

- Subsystems
- Databases
- Infrastructure
- Interfaces
- APIs
- Micro services

Integration Testing

Typical defects and failures for CIT include :

- Incorrect data, missing data, or incorrect data encoding
- Incorrect sequencing or timing of interface calls
- Interface mismatch
- Failures in communication between components
- Unhandled or improperly handled communication failures between components
- Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components

Integration Testing

Typical defects and failures for SIT include:

- Inconsistent message structures between systems
- Incorrect data, missing data, or incorrect data encoding
- Interface mismatch
- Failures in communication between systems
- Unhandled or improperly handled communication failures between systems
- Incorrect assumptions about the meaning, units, or boundaries of the data being passed between systems
- Failure to comply with mandatory security regulations

Specific approaches and responsibilities in Component Testing :
CIT and SIT should concentrate on the integration itself.
Example for CIT and how it differs from Component Testing :
CIT : if integrating module A with module B, tests should focus on the communication between the modules
Component Testing : tests focus on the functionality of the individual modules.

Example for SIT and how it differs from System Testing :
SIT : If integrating system X with system Y, tests should focus on the communication between the systems
System Testing : tests focus on the functionality of the individual systems.

CIT is often the responsibility of developers. SIT is generally the responsibility of testers. Ideally, testers performing SIT should understand the system architecture, and should have influenced integration planning. If integration tests and the integration strategy are planned before components or systems are built, those components or systems can be built in the order required for most efficient testing.
Systematic integration strategies may be based on the system architecture (e.g., top-down and bottom-up), functional tasks, transaction processing sequences, or some other aspect of the system or components.
In order to simplify defect isolation and detect defects early, integration should normally be incremental (i.e., a small number of additional components or systems at a time) rather than “big bang” (i.e., integrating all components or systems in one single step).
A risk analysis of the most complex interfaces can help to focus the integration testing. The greater the scope of integration, the more difficult it becomes to isolate defects to a specific component or system, which may lead to increased risk and additional time for troubleshooting. This is one reason that continuous integration (CI) , where software is integrated on a component-by-component basis (i.e., functional integration), has become common practice. Such continuous integration often includes automated regression testing, ideally at multiple test levels.

Types of Integration testing

Modules are integrated by two ways.

1. **Non-incremental Testing (Big Bang Testing)**
   - Each Module is tested independently and at the end, all modules are combined to form a application
1. **Incremental Module Testing.**
   - There are two types by which incremental module testing is achieved.
     - **Top down Approach :** Firstly top module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested and it is called as Stub.
     - **Bottom up Approach :** Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested. This continues till top most module is added to rest all and tested and it is called as Driver.

Top-Down Testing :
Firstly top module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested.

The main control module is used as a test driver
Stubs are substituted for all components directly subordinate to the main control module
Later, the subordinate stubs are replaced by actual components

Disadvantages:
Many tests are delayed until stubs are replaced by actual modules.
Time taken to develop stubs to perform the functions of the actual modules.

Advantage : Fast

Bottom-up Testing :

Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested.

This continues till top most module is added to rest all and tested.

Low-level components are combined into clusters (builds) that perform a specific sub function

A driver is written to coordinate test case input and output

Drivers are removed and clusters are combined moving upward in the program structure

2.2.3 System Testing

- System testing focuses on the behavior and capabilities of a whole system or product, often considering the end-to-end tasks the system can perform and the non-functional behaviors it exhibits while performing those tasks.
- Test the software in the real environment in which it is to operate. (hardware, people, information, etc.)
- Observe how the system performs in its target environment, for example in terms of speed, with volumes of data, many users, all making multiple requests.
- Test how secure the system is and how can the system recover if some fault is encountered in the middle of procession.
- System Testing, by definition, is impossible if the project has not produced a written set of measurable objectives for its product.

System testing

- Objectives of System testing include:
  - Reducing risk
  - Verifying whether the functional and non-functional behaviors of the system are as designed and specified
  - Validating that the system is complete and will work as expected
  - Building confidence in the quality of the system as a whole
  - Finding defects
  - Preventing defects from escaping to higher test levels or production

For certain systems, verifying data quality may be an objective. As with component testing and integration testing, in some cases automated system regression tests provide confidence that changes have not broken existing features or end-to-end capabilities.
System testing often produces information that is used by stakeholders to make release decisions.
System testing may also satisfy legal or regulatory requirements or standards.
The test environment should ideally correspond to the final target or production environment.

System Testing

Test Basis : Test basis for System testing include:

- System and software requirement specifications (functional and non-functional)
- Risk analysis reports
- Use cases
- Epics and user stories
- Models of system behavior
- State diagrams
- System and user manuals

Test objects : Typical test objects for System testing include:

- Applications
- Hardware/software systems
- Operating systems
- System under test (SUT)
- System configuration and configuration data

System Testing

Typical defects and failures :

- Typical defects and failures for System Testing include :
  - Incorrect calculations
  - Incorrect or unexpected system functional or non-functional behavior
  - Incorrect control and/or data flows within the system
  - Failure to properly and completely carry out end-to-end functional tasks
  - Failure of the system to work properly in the production environment(s)
  - Failure of the system to work as described in system and user manuals

Specific approaches and responsibilities in Component Testing :
System testing should focus on the overall, end-to-end behavior of the system as a whole, both functional and non-functional. System testing should use the most appropriate techniques (see chapter 4) for the aspect(s) of the system to be tested. For example, a decision table may be created to verify whether functional behavior is as described in business rules.
Independent testers typically carry out system testing. Defects in specifications (e.g., missing user stories, incorrectly stated business requirements, etc.) can lead to a lack of understanding of, or disagreements about, expected system behavior. Such situations can cause false positives and false negatives, which waste time and reduce defect detection effectiveness, respectively.
Early involvement of testers in user story refinement or static testing activities, such as reviews, helps to reduce the incidence of such situations.

Types of System Testing

- Functional Testing
- Performance Testing
- Volume Testing
- Load Testing
- Stress Testing
- Security Testing
- Web Security Testing
- Localization Testing
- Usability Testing
- Recovery Testing
- Documentation Testing
- Configuration Testing
- Installation Testing
- User Acceptance Testing
- Testing related to Changes : Re-Testing and Regression Testing
- Re-testing (Confirmation Testing)
- Regression Testing
- Exploratory Testing
- Maintenance Testing

Functional Testing

The main objective of functional testing is to verify that each function of the software application / system operates in accordance with the written requirement specifications.

It is a black-box process :

- Is not concerned about the actual code
- Focus is on validating features
- Uses external interfaces, including Application programming interfaces (APIs), Graphical user interfaces (GUIs) and Command line interfaces (CLIs)

Testing functionality can be done from two perspectives :

1. Business-process-based testing uses knowledge of the business processes
2. Requirements-based testing uses a specification of the functional requirements for the system as the basis for designing tests

Performance Testing

Performance

- Performance is the behavior of the system w.r.t. goals for time, space, cost and reliability

Performance objectives:

- **Throughput :** The number of tasks completed per unit time. Indicates how much work has been done within an interval
- **Response time :** The time elapsed during input arrival and output delivery
- **Utilization :** The percentage of time a component (CPU, Channel, storage, file server) is busy

Objectives of performance testing :

- To devise test case that attempts to show that the program does not satisfy its performance objectives.
- To ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.
- To test response time and reliability by increased user traffic.
- To identify which components are responsible for performance degradation and what usage characteristics cause degradation to occur.

Volume Testing

This testing is subjecting the program to heavy volumes of data. For e.g.

- A compiler would be fed a large source program to compile
- An operating systems job queue would be filled to full capacity
- A file system would be fed with enough data to cause the program to switch from one volume to another.

Load Testing

Volume testing creates a real-life end user pressure for the target software. This tests how the software acts when numerous end users access it concurrently. For e.g.

- Downloading a sequence of huge files from the web
- Giving lots of work to a printer in a line

Stress Testing

- Stress testing involves subjecting the program to heavy loads or stresses.
- The idea is to try to “break” the system.
- That is, we want to see what happens when the system is pushed beyond design limits.
- It is not same as volume testing.
- A heavy stress is a peak volume of data encounters over a short time.
- In Stress testing a considerable load is generated as quickly as possible in order to stress the application and analyze the maximum limit of concurrent users the application can support.

Stress Testing(Cont.)

Stress tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

Example :

- Generate 5 interrupts when the average rate is 2 or 3
- Increase input data rate
- Test cases that require max. memory

Stress Tests should answer the following questions

- Does the system degrade gently or does the server shut down
- Are appropriate messages displayed ? E.g. Server not available
- Are transactions lost as capacity is exceeded
- Are certain functions discontinued as capacity reaches the 80 or 90 percent level

Security Testing

Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.

Security testing is the process of executing test cases that subvert the program's security checks.

Example :

- One tries to break the operating systems memory protection mechanisms
- One tries to subvert the DBMS's data security mechanisms
- The role of the developer is to make penetration cost more than the value of the information that will be obtained

Any computer based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities : hackers who attempt to penetrate systems for sport, disgruntled employees who attempt to penetrate for revenge, dishonest individuals who attempt to penetrate for illicit personal gain.

Firewalls – a filtering mechanism that is a combination of hardware and software that examines each incoming packet of information to ensure that it is coming from a legitimate source, blocking any data that are suspect.
Authentication – a verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.
Encryption – Protect sensitive data by modifying it in a way that makes it impossible to read by those with malicious intent.
Authorization – a filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes. (e.g. Userid , Passwords)

Web Security Testing

Web application security is a branch of Information Security that deals specifically with security of web applications.

It provides a strategic approach in identifying, analyzing and building a secure web applications.

It is performed by Web Application Security Assessment.

Why, Web Application Security?

1. Top breaches/fraud in recent times
2. Heartland Payment Systems - In January 2009 attackers were able to steal more than 130,000,000 credit card records
3. Virginia State Prescription Monitoring Program Records - Hackers stole 8.3 million records, erased the originals and created an encrypted backup of VPMP's database
4. Terrorists intercept US Drone unencrypted Video Feeds - Islamic terrorists have been able to hack into CIA state-of-the-art Predator drones with the help of just a 25.95 dollar off-the-shelf software, raising fears of remote control operated unmanned crafts being taken over and used against British and American targets.
5. Phishing attacks on banking sites – ICICI, SBI etc.,

Localization Testing

Localization translates the product UI and occasionally changes some settings to make it suitable for another region.

The test effort during localization testing focuses on

- Areas affected during localization, UI and content
- Culture/locale-specific, language specific and region specific areas

The goal of globalization testing is to detect potential problems in application design that could inhibit globalization. It makes sure that the code can handle all international support without breaking the functionality that would cause either data loss or display problems.

Automated testing is not an effective solution for localization/globalization testing because the default mouse click position changes with the language and also the text in the recordings don’t obviously match for all languages.

Usability Testing

Usability is

- The effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment ISO 9241-11
- Effective–- Accomplishes user's goal
- Efficient-- Accomplishes the goal quickly
- Satisfaction–- User enjoys the experience

Test Categories and objectives

- Interactivity (Pull down menus, buttons)
- Layout
- Readability
- Aesthetics
- Display characteristics
- Time sensitivity
- Personalization

Usability testing can be formal, informal or heuristic based on the needs and the availability.

There are many frameworks formulated for usability testing like NIST has formulated a CIF (Common Industry Format) for reporting the findings of the test.

Interactivity – Are interaction mechanisms(pull down menus ,pointers) easy to understand

Layout – Are navigation mechanisms, content and functions placed in a manner that allows the user to find them quickly?

Readability – Is text well written and understandable ? Are graphic representation easy to understand?

Aesthetics – Do layout, color, typeface and related characteristics lead to ease of use? Do users feel comfortable with the look and feel.

Display Characteristics – optimal use of screen size and resolution.

Time sensitivity – Can important features, functions and content be used or acquired in a timely manner

Personalization – Does the web application tailor itself to the specific needs of different user categories or individual users?

Usability Testing (Cont.)

Using specialized Test Labs a rigorous testing process is conducted to get quantitative and qualitative data on the effectiveness of user interfaces

Representative or actual users are asked to perform several key tasks under close observation, both by live observers and through video recording

During and at the end of the session, users evaluate the product based on their experiences

Trainer notes

Recovery Testing

A system test that forces the software to fail in variety of ways, checks performed

- recovery is automatic (performed by the system itself)
- reinitialization
- check pointing mechanisms
- data recovery
- restarts are evaluated for correctness

This test confirms that the program recovers from expected or unexpected events. Events can include shortage of disk space, unexpected loss of communication

It Confirms that the program recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.

Documentation Testing

This testing is done to ensure the validity and usability of the documentation

This includes user Manuals, Help Screens, Installation and Release Notes

Purpose is to find out whether documentation matches the product and vice versa

Well-tested manual helps to train users and support staff faster

Generally, Technical Writers work to tight schedules, which often does not include documentation testing because there is no time. Besides, no one wants to take the risk of causing a rewrite or correcting product design and not shipping on schedule. Bad documentation has a ripple effect on the number of users it impacts such as Product Development, Training, and Customer Support.

Configuration Testing

Attempts to uncover errors that are specific to a particular client or server environment.

Create a cross reference matrix defining all probable operating systems, browsers, hardware platforms and communication protocols.

Test to uncover errors associated with each possible configuration

Configuration Test
Analyze system behaviour:
in various hardware and software configurations specified in the requirements
sometimes systems are built in various configurations for different users
for instance, a minimal system may serve a single user, other configurations for additional users.

Installation Testing

Installer is the first contact a user has with a new software!!!

Installation testing is required to ensure:

- Application is getting installed properly
- New program that is installed is working as desired
- Old programs are not hampered
- System stability is maintained
- System integrity is not compromised

2.2.4 Acceptance testing

- Acceptance testing, like system testing, typically focuses on the behavior and capabilities of a whole system or product
- Objectives of Acceptance testing include:
  - Establishing confidence in the quality of the system as a whole
  - Validating that the system is complete and will work as expected
  - Verifying that functional and non-functional behaviors of the system are as specified

Acceptance testing may produce information to assess the system's readiness for deployment and use by the customer (end-user). Defects may be found during acceptance testing, but finding defects is often not an objective, and finding a significant number of defects during acceptance testing may in some cases be considered a major project risk. Acceptance testing may also satisfy legal or regulatory requirements or standards.

Forms of Acceptance testing

Common forms of acceptance testing include the following:

- User Acceptance testing (UAT)
- Operational Acceptance testing (OAT)
- Contractual and Regulatory Acceptance testing
- Alpha and Beta testing.

User Acceptance Testing (UAT) - Focuses mainly on the functionality thereby validating the fitness-for-use and is performed by the users and application managers Operational Acceptance Test / Production Acceptance Test - Validates whether the system meets the requirements for operation and may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities
Contract Acceptance Testing - Is performed against a contract's acceptance criteria for producing custom-developed software
Compliance acceptance testing / Regulation Acceptance Testing - Is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.
Alpha Testing – It is performed at the developer's site by a cross-section of potential users and members of the developer's organization
Beta Testing - It is performed by a cross-section of users who install it and use it under real-world working conditions

User Acceptance Testing (UAT)

- A test executed by the end user(s) is typically focused on validating the fitness for use of the system by intended users in a real or simulated operational environment. The main objective is building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost, and risk.

- Usually carried out by the end user to test whether or not the right system has been created

Operational Acceptance Testing (OAT)

- Acceptance testing of the system by operations or systems administration staff is usually performed in a (simulated) production environment.
- The tests focus on operational aspects, and may include:
  - Testing of backup and restore
  - Installing, uninstalling and upgrading
  - Disaster recovery
  - User management
  - Maintenance tasks
  - Data load and migration tasks
  - Checks for security vulnerabilities
  - Performance testing

The main objective of OAT is building confidence that the operators or system administrators can keep the system working properly for the users in the operational environment, even under exceptional or difficult conditions

Contractual and Regulatory Testing

- **Contractual** acceptance testing is performed against a contract's acceptance criteria for producing custom-developed software. Acceptance criteria should be defined when the parties agree to the contract. Contractual acceptance testing is often performed by users or by independent testers.

- **Regulatory** acceptance testing is performed against any regulations that must be adhered to, such as government, legal, or safety regulations. Regulatory acceptance testing is often performed by users or by independent testers, sometimes with the results being witnessed or audited by regulatory agencies.

The main objective of contractual and regulatory acceptance testing is building confidence that contractual or regulatory compliance has been achieved

Alpha and Beta Testing

- Alpha and beta testing are typically used by developers of commercial off-the-shelf (COTS) software who want to get feedback from potential or existing users, customers, and/or operators before the software product is put on the market.
- Alpha testing is performed at the developing organization's site, not by the development team, but by potential or existing customers, and/or operators or an independent test team.
- Beta testing is performed by potential or existing customers, and/or operators at their own locations. Beta testing may come after alpha testing, or may occur without any preceding alpha testing having occurred.

One objective of alpha and beta testing is building confidence among potential or existing customers, and/or operators that they can use the system under normal, everyday conditions, and in the operational environment(s) to achieve their objectives with minimum difficulty, cost, and risk.
Another objective may be the detection of defects related to the conditions and environment(s) in which the system will be used, especially when those conditions and environment(s) are difficult to replicate by the development team.

Alpha and Beta Testing

Test Basis

Examples of work products that can be used as a test basis for any form of acceptance testing include:

- Business processes
- User or business requirements
- Regulations, legal contracts and standards
- Use cases
- System requirements
- System or user documentation
- Installation procedures
- Risk analysis reports

In addition, as a test basis for deriving test cases for operational acceptance testing, one or more of the following work products can be used:

- Backup and restore procedures
- Disaster recovery procedures
- Non-functional requirements
- Operations documentation
- Deployment and installation instructions
- Performance targets
- Database packages
- Security standards or regulations

Alpha and Beta Testing

Test Objects

Typical test objects for any form of acceptance testing include:

- System under test
- System configuration and configuration data
- Business processes for a fully integrated system
- Recovery systems and hot sites (for business continuity and disaster recovery testing)
- Operational and maintenance processes
- Forms
- Reports
- Existing and converted production data

Alpha and Beta Testing

Typical Defects and Failures

Typical defects for any form of acceptance testing include:

- System workflows do not meet business or user requirements
- Business rules are not implemented correctly
- System does not satisfy contractual or regulatory requirements
- Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform

Specific approaches and responsibilities

Acceptance testing is often the responsibility of the customers, business users, product owners, or operators of a system, and other stakeholders may be involved as well.

Acceptance testing is often thought of as the last test level in a sequential development lifecycle, but it may also occur at other times, for example:

- Acceptance testing of a COTS software product may occur when it is installed or integrated
- Acceptance testing of a new functional enhancement may occur before system testing

In iterative development, project teams can employ various forms of acceptance testing during and at the end of each iteration, such as those focused on verifying a new feature against its acceptance criteria and those focused on validating that a new feature satisfies the users' needs.

User acceptance tests, operational acceptance tests, regulatory acceptance tests, and contractual acceptance tests, alpha and beta tests may occur, either at the close of each iteration, after the completion of each iteration, or after a series of iterations

2.3 Test Types

A test type is a group of test activities aimed at testing specific characteristics of a software system based on specific test objectives such as :

- Evaluating functional quality characteristics, such as completeness, correctness, and appropriateness
- Evaluating non-functional quality characteristics, such as reliability, performance efficiency, security, compatibility, and usability
- Evaluating whether the structure or architecture of the component or system is correct, complete, and as specified
- Evaluating the effects of changes, such as confirming that defects have been fixed (confirmation testing) and looking for unintended changes in behavior resulting from software or environment changes (regression testing)

2.3.1 Functional Testing

- Functional testing of a system involves tests that evaluate functional requirements such as business requirements specifications, epics, user stories, use cases, or functional specifications, or they may be undocumented.
- The functions are "what" the system should do
- Functional tests should be performed at all test levels though the focus is different at each level.
- Black-box techniques may be used to derive test conditions and test cases for the functionality of the component or system

The thoroughness of functional testing can be measured through functional coverage. Functional coverage is the extent to which some type of functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being covered.

For example, using traceability between tests and functional requirements, the percentage of these requirements which are addressed by testing can be calculated, potentially identifying coverage gaps.

Functional test design and execution may involve special skills or knowledge, such as knowledge of the particular business problem the software solves (e.g., geological modelling software for the oil and gas industries) or the particular role the software serves (e.g., computer gaming software that provides interactive entertainment).

2.3.2 Non-Functional Testing

- Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security.
- Non-functional testing is the testing of "how well" the system behaves.
- non-functional testing can and often should be performed at all test levels, and done as early as possible.
- Black-box techniques may be used to derive test conditions and test cases even for nonfunctional testing too. For example, boundary value analysis can be used to define the stress conditions for performance tests.

The thoroughness of non-functional testing can be measured through non-functional coverage. Nonfunctional coverage is the extent to which some type of non-functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being covered.

For example, using traceability between tests and supported devices for a mobile application, the percentage of devices which are addressed by compatibility testing can be calculated, potentially identifying coverage gaps

Non-functional test design and execution may involve special skills or knowledge, such as knowledge of the inherent weaknesses of a design or technology (e.g., security vulnerabilities associated with particular programming languages) or the particular user base (e.g., the personas of users of healthcare facility management systems).

2.3.3 White-box Testing

- White-box testing derives tests based on the system's internal structure or implementation. Internal structure may include code, architecture, work flows, and/or data flows within the system.

- White-box testing derives tests based on the system's internal structure or implementation. Internal structure may include code, architecture, work flows, and/or data flows within the system.

At the component testing level, code coverage is based on the percentage of component code that has been tested, and may be measured in terms of different aspects of code (coverage items) such as the percentage of executable statements tested in the component, or the percentage of decision outcomes tested. These types of coverage are collectively called code coverage.

At the component integration testing level, white-box testing may be based on the architecture of the system, such as interfaces between components, and structural coverage may be measured in terms of the percentage of interfaces exercised by tests.

2.3.4 Change-related Testing

- When changes are made to a system, either to correct a defect or because of new or changing functionality, testing should be done to confirm that the changes have corrected the defect or implemented the functionality correctly, and have not caused any unforeseen adverse consequences.
  - Confirmation testing
  - Regression testing
- Confirmation testing and regression testing are performed at all test levels.
  - Especially in iterative and incremental development lifecycles (e.g., Agile), new features, changes to existing features, and code refactoring results in frequent changes to the code, which requires change-related testing.
  - Due to the evolving nature of the system (e.g. IoT system), confirmation and regression testing are very important.

Re-testing (Confirmation Testing)

- After a defect is fixed, the software is tested by re-running all test cases that failed due to the defect, within same environment, with same inputs and same preconditions on the new software version.
- The software may also be tested with new tests if, for instance, the defect was missing functionality.
- At the very least, the steps to reproduce the failure(s) caused by the defect must be re-executed on the new software version.
- The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed

Regression Testing

- It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems.
- Changes may include changes to the environment, such as a new version of an operating system or database management system.
- Such unintended side-effects are called regressions.
- Regression testing involves running tests to detect such unintended side-effects

2.3.5 Test Types and Test Levels
Example: Banking Application

It is possible to perform any of the test types at any test level.

Examples of Functional Tests :

- For CT, tests are designed based on how a component should calculate compound interest.
- For CIT, tests are designed based on how account information captured at the user interface is passed to the business logic.
- For ST, tests are designed based on how account holders can apply for a line of credit on their checking accounts.
- For SIT, tests are designed based on how the system uses an external micro service to check an account holder’s credit score.
- For UAT, tests are designed based on how the banker handles approving or declining a credit application.

2.3.5 Test Types and Test Levels (Cont..)

Example: Banking Application

Examples of Non-Functional Tests :

- For CT, performance tests are designed to evaluate the number of CPU cycles required to perform a complex total interest calculation
- For CIT, security tests are designed for buffer overflow vulnerabilities due to data passed from the user interface to the business logic.
- For ST, portability tests are designed to check whether the presentation layer works on all supported browsers and mobile devices.
- For SIT, reliability tests are designed to evaluate system robustness if the credit score micro service fails to respond.
- For UAT, usability tests are designed to evaluate the accessibility of the banker’s credit processing interface for people with disabilities.

2.3.5 Test Types and Test Levels (Cont..)

Example: Banking Application

Examples of White-box Tests :

- For CT, performance tests are designed to achieve complete statement and decision coverage for all components that perform financial calculations.
- For CIT, security tests are designed to exercise how each screen in the browser interface passes data to the next screen and to the business logic.
- For ST, tests are designed to cover sequences of web pages that can occur during a credit line application.
- For SIT, tests are designed to exercise all possible inquiry types sent to the credit score microservice.
- For UAT, tests are designed to cover all supported financial data file structures and value ranges for bank-to-bank transfers.

2.3.5 Test Types and Test Levels (Cont..)

Example: Banking Application

Examples of change-related Tests :

- For CT, automated regression tests are built for each component and included within the continuous integration framework.
- For CIT, tests are designed to confirm fixes to interface-related defects as the fixes are checked into the code repository.
- For ST, all tests for a given workflow are re-executed if any screen on that workflow changes.
- For SIT, tests of the application interacting with the credit scoring micro service are re-executed daily as part of continuous deployment of that micro service.
- For UAT, all previously-failed tests are re-executed after a defect found in acceptance testing is fixed.

Though this section provides examples of every test type across every level, it is not necessary, for all software, to have every test type represented across every level. However, it is important to run applicable test types at each level, especially the earliest level where the test type occurs.

2.4 Maintenance Testing

- Once deployed to production environments, software and systems need to be maintained.
- Changes of various sorts are almost inevitable in delivered software and systems, either to fix defects discovered in operational use, to add new functionality, or to delete or alter already-delivered functionality.
- Maintenance is also needed to preserve or improve non-functional quality characteristics of the component or system over its lifetime, especially performance efficiency, compatibility, reliability, security, compatibility, and portability.

2.4 Maintenance Testing (cont..)

- When any changes are made as part of maintenance, maintenance testing should be performed, both to evaluate the success with which the changes were made and to check for possible side-effects (e.g., regressions) in parts of the system that remain unchanged.
- Maintenance can involve planned releases and unplanned releases (hot fixes).
- Maintenance testing is required at multiple test levels, using various test types, based on its scope. The scope of maintenance testing depends on:
  - The degree of risk of the change, for example, the degree to which the changed area of software communicates with other components or systems
  - The size of the existing system
  - The size of the change

Maintenance testing focuses on testing the changes to the system, as well as testing unchanged parts that might have been affected by the changes.

2.4.1 Triggers for Maintenance

There are several reasons (triggers) why maintenance testing takes place, both for planned and unplanned changes :

- Modification, such as planned enhancements (e.g., release-based), corrective and emergency changes, changes of the operational environment (such as planned operating system or database upgrades), upgrades of COTS software, and patches for defects and vulnerabilities
- Migration, such as from one platform to another, which can require operational tests of the new environment as well as of the changed software, or tests of data conversion when data from another application will be migrated into the system being maintained
- Retirement, such as when an application reaches the end of its life

When an application or system is retired, this can require testing of data migration or archiving if long data-retention periods are required.
Testing restore/retrieve procedures after archiving for long retention periods may also be needed.
In addition regression testing may be needed to ensure that any functionality that remains in service still works.

For Internet of Things systems, maintenance testing may be triggered by the introduction of completely new or modified things, such as hardware devices and software services, into the overall system.
The maintenance testing for such systems places particular emphasis on integration testing at different levels (e.g., network level, application level) and on security aspects, in particular those relating to personal data.

2.4.2 Impact Analysis for Maintenance

Impact analysis may be done before a change is made, to help decide if the change should be made, based on the potential consequences in other areas of the system.

Impact analysis can be difficult if:

- Specifications (e.g., business requirements, user stories, architecture) are out of date or missing
- Test cases are not documented or are out of date
- Bi-directional traceability between tests and the test basis has not been maintained
- Tool support is weak or non-existent
- The people involved do not have domain and/or system knowledge
- Insufficient attention has been paid to the software's maintainability during development

Impact analysis evaluates the changes that were made for a maintenance release to identify the intended consequences as well as expected and possible side effects of a change, and to identify the areas in the system that will be affected by the change. Impact analysis can also help to identify the impact of a change on existing tests. The side effects and affected areas in the system need to be tested for regressions, possibly after updating any existing tests affected by the change.

2.5 Test Case Terminologies

Pre Condition

- Environmental and state which must be fulfilled before the component/unit can be executed with a particular input value.

Test Analysis

- is a process for deriving test information by viewing the Test Basis
- For testing, test basis is used to derive what could be tested

Test basis includes whatever the test are based on such as System Requirement

- A Technical specification
- The code itself (for structural testing)
- A business process

Test Condition

- It is a set of rules under which a tester will determine if a requirement is partially or fully satisfied
- One test condition will have multiple test cases

Test Case Terminologies (cont.)

Test Scenario

- It is an end-to-end flow of a combination of test conditions & test cases integrated in a logical sequence, covering a business processes
- This clearly states what needs to be tested
- One test condition will have multiple test cases

Test Procedure (Test Steps)

- A detailed description of steps to execute the test

Test Data/Input

- Inputs & its combinations/variables used

Expected Output

- This is the expected output for any test case or any scenario

Actual Output

- This is the actual result which occurs after executing the test case

Test Result/Status

- Pass / Fail – If the program works as given in the specification, it is said to Pass otherwise Fail.
- Failed test cases may lead to code rework

Other Terminologies

Test Suite – A set of individual test cases/scenarios that are executed as a package, in a particular sequence and to test a particular aspect

- E.g. Test Suite for a GUI or Test Suite for functionality

Test Cycle – A test cycle consists of a series of test suites which comprises a complete execution set from the initial setup to the test environment through reporting and clean up.

- E.g. Integration test cycle / regression test cycle

Test Suite – Test suite is set of all test cases. Suites are usually related by the area of the application they exercise or by their priority or content.
For E.g. When you Login to the screen, some functionalities like validating user name, password with different invalid inputs can act as Test suites.
E.g. In case of ATM machine, deposit, withdraw, balance check are separate test suites that carry out different test cases

Test Cycle – It's a combination of series of test suites.
For E.g. The Test Cycle for the same application is combination of multiple Test Suites like, Functional validations, Database validations, GUI validations, etc. form the Test Cycle.
E.g. Combination of all test suites like deposit, withdraw, balance check in case of ATM machine, forms a complete cycle

A good Test Case

Has a high probability of detecting error(s)
Test cases help us discover information
Maximize bug count
Help managers make ship / no-ship decisions
Minimize technical support costs
Assess conformance to specification
Verify correctness of the product
Minimize safety-related lawsuit risk
Find safe scenarios for use of the product
Assure quality

Features of a good Test Case:

Detecting defects. This is the classic objective of testing. A test is run in order to trigger failures that expose defects. Generally, we look for defects in all interesting parts of the product.

Maximize bug count. The distinction between this and “find defects” is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high-risk features, if this is the way to find the most bugs in the time available.

Help managers make ship / no-ship decisions. Managers are typically concerned with risk in the field. They want to know about coverage (maybe not the simplistic code coverage statistics, but some indicators of how much of the product has been addressed and how much is left), and how important the known problems are. Problems that appear significant on paper but will not lead to customer dissatisfaction are probably not relevant to the ship decision.

Minimize technical support costs. Working in conjunction with a technical support or help desk group, the test team identifies the issues that lead to calls for support. These are often peripherally related to the product under test--for example, getting the product to work with a specific printer or to import data successfully from a third party database might prevent more calls than a low-frequency, data-corrupting crash.

2.6 Test data

- An application is built for a business purpose. We input data and there is a corresponding output. While an application is being tested we need to use dummy data to simulate the business workflows. This is called test data.

- The test data may be any kind of input to application, any kind of file that is loaded by the application or entries read from the database tables. It may be in any format like xml test data, stand alone variables, SQL test data etc.

Properties of Good Test Data

Realistic – accurate in context of real life

- E.g. Age of a student giving graduation exam is at least 18

Practically valid – data related to business logic

- E.g. Age of a student giving graduation exam is at least 18 says that 60 years is also valid input but practically the age of a graduate student cannot be 60

Cover varied scenarios

- E.g. Don’t just consider the scenario of only regular students but also consider the irregular students, also the students who are giving a re-attempt, etc.

Exceptional data

- E.g. There may be few students who are physically handicapped must also be considered for attempting the exam

Summary

In this lesson, you have learnt:

- Verification refers to a set of activities which ensures that software correctly implements a specific function.
- Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
- Different testing phases are
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing

Review Question

Question 1: _____ is a Quality improvement process

Question 2: To test a function, the programmer has to write a _____, which calls the function to be tested and passes it test data

Question 3: Volume tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

- True/False

Question 4: Acceptance testing is not a responsibility of the Developing Organization

- True/False

Question 5: Difference between re-testing & regression testing is :

- re-testing is running a test again; regression testing looks for unexpected side effects
- re-testing looks for unexpected side effects; regression testing is repeating

Review Question: Match the Following

1. Beta testing	A. Volume testing
2. Response time	B. Exploratory testing
3. Aesthetics	C. Acceptance testing
	D. Documentation testing
	E. Performance testing
	F. Usability testing

Review Question: Match the Following

1. Economics of limiting	A. Driving
2. Testing	B. Exhaustive testing
3. A good test case	C. Limiting
4. Use every possible input condition as a test case	D. Test cycle
	E. Comparing outputs with specified or intended
	F. Maximize bug count