# “First Actual Case of Bug being found”

|  |
| --- |
|  |

Inventors and engineers had been talking about bugs for more than a century before the moth in the relay incident. Even Thomas Edison used the word. Here's an extract of a letter he wrote in 1878 to Theodore Puskas, as cited in The Yale Book of Quotations (2006):

'Bugs' -- as such little faults and difficulties are called -- show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.

# The Cost of Software Failure

In the Therac-25 radiation therapy machine, an ‘arithmetic overflow’ sometimes occurred during automatic safety checks. - Death of 4 patients

The defect in a stock trading software pushed the price of the stocks up, resulting in spectacular losses for the trading firm when it had to sell the overvalued stocks back into the market at a lower price. - $440 million

In the 2012 US elections, touchscreen errors automatically changing the vote from one candidate to another and not allowing voters to reselect or correct the error. - Integrity of the results

# Software Quality Views

(Kaik & Tripathy, 2008)

Philosophical – we know quality when we see it

User – does the product do what it was created to do?

Manufacturing – does the product meet its specifications?

Product – is it flexible? Is it maintainable? Is it testable?

Value-based – is the product worth its value?

# The Pendulum Perspective of Quality

(Desikan & Ramesh, 2006)

Quality Control (one end)

* Defect detection and correction

Quality Assurance (the other end)

* Defect prevention

Companies tend to swing from one extreme to the other

Note:

* Defect detection and correction in earlier phases of the development cycle acts as defect prevention in later phases.
* It is far cheaper to detect and correct defects in earlier phases than in later ones.

# Verification and Validation

(Collofello, 1988)

Validation refers to the process of evaluating software at the end of its development to insure that it is free from failures and complies with its requirements.

Verification refers to the process of determining whether or not the products of a given phase of a software development process fulfill the requirements established during the previous phase.

# Software Defect Terminology

**Mistake**: A human action that produces an incorrect step (a fault).

**Fault (or Defect)**: An incorrect step, process, or data definition. The outgrowth of a mistake.

**Error**: The amount by which the result is incorrect. That is, the degree of the failure.

**Failure**: An incorrect result. The manifestation of a fault.

## Year 2038 problem

**Mistake**: deciding to use a signed 32-bit integer to store number of seconds since 00:00:00 UTC, Jan 1, 1970

**Fault**: Using the signed 32-bit integer

**Error**: No error yet

**Failure**: we’ll see!

# Software Development Phases

Requirements - Requirements Verification

Design - Design Verification

Coding - Code Verification

Testing - Requirements Validation

Deployment and Maintenance – Requirements Validation

# V-Model V&V

|  |
| --- |
|  |

# Six Essentials of Software Testing

1. The quality of the test process determines the success of the test effort.
2. Prevent defect migration by using early life-cycle testing techniques.
3. The time for software testing tools is now.
4. A real person must take responsibility for improving the testing process.
5. Testing is a professional discipline requiring trained, skilled people.
6. Cultivate a positive team attitude of creative destruction.

# A Few Software Quality Attributes

Correctness – does the software satisfy all requirements of the system?

Reliability – does the software always give correct results?

Efficiency – does the software use minimal resources to produce the results?

Integrity – can the software be trusted? Is it complete?

Usability – can the software be conveniently used by the intended audience?

Maintainability – can the software be conveniently modified?

Interoperability – does the software work well with other systems

# V&V Non-Agile Process

1. Formulate and baseline a test plan
2. Design test case specifications
3. Update the traceability matrix
4. Identify which tests are to be automated
5. Develop and baseline test cases
6. Execute test cases and keep traceability matrix current
7. Collect and analyze metrics
8. Prepare test summary report
9. Recommend product release criteria

# Test First Development

1. Create a test
2. Run the tests
3. If a test fails, make a little change, go to 2
4. Refactor
5. If more to develop, go to 1

TDD – Testing drives the design and implementation of the software application

ADD – TDD but the tests are written from the user’s point of view

BDD – TDD with semi-formal techniques used to define the behavioral specifications

# Test Plan

* Identification of what needs to be tested
* Specification of how the testing will be performed
* Identification of needed resources
* Testing activity breakdown and scheduling
* Risk analysis

# Test Case Development

1. Identify the purpose of the test
2. Identify the system under test (SUT)
3. Identify the environment that needs to be set up for running the test case
4. Identify the input data to be used for the test case
5. Identify the steps to be followed to execute the test
6. Identify the expected results that are considered to be correct.
7. Identify a step to compare the actual results to the expected results
8. Identify any relationships between this test and other tests

# Traceability Matrix

A traceability matrix is a document, usually in the form of a table, which correlates baselined items that require a many-to-many relationship to determine the completeness of the relationship.

|  |
| --- |
|  |

# To Automate or not to Automate

Criteria

* How repetitive are the tests?
* How much effort is involved in automating?
* How much manual intervention is needed for the test?
* How much does the automation tool cost?

# Writing Test Cases

Automated

* Writing of the scripts in the automation language

Manual

* Writing detailed step-by-step instructions for executing the test and validating the results

Should also have change history documentation

1. What was the change?
2. Why was the change necessary?
3. Who made the change?
4. When was the change made?
5. How was the change implemented?
6. Other files affected by the change

# Test Case Execution

Updating of the defect repository

* Defect Repository:
  + The primary vehicle of communication between test team and development team
  + Contains information about all defects

# Collecting and Analyzing Metrics

**Project Metrics**: indicate how the project is planned and executed

* Effort Variance
* Schedule Variance
* Effort distribution across phases

**Progress Metrics**: tracks how different activities of the project are progressing

* Defect find rate
* Defect fix rate
* Outstanding defects rate
* Priority outstanding rate
* Defect trend
* Defect classification trend
* Weighted defects trend
* Defect cause trend
* Component-wise defect distribution
* Defect density and defect removal rate
* Age analysis of outstanding defects
* Introduced and reopened defects trend

**Productivity Metrics**: helps in planning and estimating test activities

* Defects per 100 hours of testing
* Test cases executed per 100 hours of testing
* Test cases developed per 100 hours of testing
* Defects per 100 test cases
* Defects per 100 failed test cases
* Test phase effectiveness
* Closed defect distribution

**Release Metrics**

* A consideration of all previous metrics

# Test Reporting

Test Incident Report

* An entry in the defect repository

Test Cycle Report

* Summary of activities during this cycle
* Defects uncovered
* Defects fixed since last cycle
* Outstanding defects

Test Summary Report

* Summary of activities during this cycle
* Variance from planned activities
* Summary of results
* Comprehensive assessment and recommendation for release

# Test Case Database

Test case

* Test case ID
* Test case name
* Test case owner
* Associated files for the test case
* Module ID

Test case run history

* Test case ID
* Run date and time
* Time taken
* Run status (success/failure)

Test case defect report

* Test case ID
* Defect ID

Defect Details

* Defect ID
* Defect status (open/closed)
* Defect classification
* Defect description
* Affected artifact(s)
* Any relevant version information
* Environmental information
* Who encountered the problem
* Date and time of defect occurrence

Defect test details

* Defect ID
* Test case ID

Fix Details

* Defect ID
* Fix details

Communication

* Test case ID
* Defect ID
* Communication details

# Defect Classification

Extreme

* Product crashes or unstable
* Needs to be fixed immediately

Critical

* Basic functionality of the product not working
* Needs to be fixed before next test cycle starts

Important

* Extended functionality of the product not working
* Does not affect the progress of testing
* Fix it before the release

Minor

* Product behaves differently
* No impact on the test team or customers
* Fix it when time permits

Cosmetic

* Minor irritant
* Need not be fixed for this release

# Defect Priority

1. [Highest] Fix the defect on highest priority; fix it before the next build
2. [High] Fix the defect on high priority before next test cycle
3. [Moderate] Fix the defect on moderate priority when time permits, before the release
4. [Low] Postpone this defect for the next release
5. [Lowest] Live with this defect

# Testing Categories

Static Analyses

* Inspections
* Walkthroughs
* Buddy checks

Dynamic Analyses

* Unit testing
* Integration testing
* System testing
* Acceptance testing
* Regression testing
* Stress testing

## Inspections

* Planned structured meeting
* All individuals prepare for it
* Team of 3 – 6 people, led by impartial moderator
* Presenter is the reader not producer

## Walkthroughs

* Planned meeting where only the presenter must prepare
* 2 – 7 people, led by producer/author
* Presenter is the producer

## Buddy Check

* Having someone else review the artifact

## Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually verified. Unit testing is often automated but it can also be done manually.

## Integration Testing

* Testing the interaction of the various modules
* Testing the interaction with other systems

## System Testing

* Testing the complete integrated system
* Testing to the specified requirements

## Acceptance Testing

* Testing from the point of view the clients and uses of the system

## Regression Testing

* Testing done during maintenance
* Ensures that enhancements or fixes does not introduce defects to existing functionality

## Stress Testing

* Testing the system beyond the limits of specified requirements or resources

# Bibliography

Collofello, J. S. (1988). Introduction to Software Verification and Validation. *SEI Curriculum Module SEI-CM-13-1.1*.

Desikan, S., & Ramesh, G. (2006). *Software Testing Principles and Practices.* Pearson Education.

Kaik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance Theory and Practice.* Wiley.