Software Testing

Unit 4

Acceptance Testing

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

PES UNIVERSITY

List of Contents

Acceptance Testing

What is it - Testing done in accordance with customer specified criteria (Test cases, scenarios, results)

- Typically Criteria is pre-defined and mutually agreed.
- Done at specific delivery instances
- Typically done under the customer controlled environment by customer designated individuals

Acceptance Testing - Importance

- Critical for business purpose. Without a defined acceptance criteria, managing the closure is difficult
- Though software should work as per requirement, it is not easy to demonstrate ALL required functionalities, under practical constraints of time and resources
  Ex: Think of all path testing for medium complex application
- Very important for software services/project companies.

Acceptance Testing - Criteria

- Typically involves specifying business functionality of some complexity
  Ex. Should do tax deduction at source during each month salary disbursement
- Most of the high priority requirements are covered in the criteria.
- Legal & Statutory requirements
- May cover some non-functional requirements also
  Ex. Should process 100 million call records in 2 hours!

Acceptance Testing - Criteria

Criteria could also be process / procedure requirements

Ex:

Test reports should show a coverage of > 85% at component level

80 staff members trained in using the data entry

All help documents must open in “Star Office”

In service contracts, it takes form of SLA – Service Level Agreement

Defects must be fixed in 25 business hours

Acceptance Testing - Criteria

Typical Test cases cover

- Critical functionality
- Most used functionality
- End-to-end scenarios
- New functionalities – during upgrade
- Legal / statutory needs
- Functionality to work on a defined corpus of data.

Acceptance Testing - Execution

- Typically happens “On-Site” after careful environment setting
- There should be ‘stand-by’ dev team to address blocking issues
- Needs a team with very good business functional knowledge and application working knowledge
- Major defects – also defined as part of criteria – break the acceptance test
  - Has to be re-started after the defect is addressed
- Careful execution documentation is a must & final reporting.

Acceptance Testing – Practical Challenges

1. Customers are wary of providing
2. Development team might omit what is not in the Acceptance Test
3. Not easy to specify – Needs good effort.
4. Most of the times vendor company defines and gets concurrence
5. Multiple iterations may be required and multiple customer representatives may be involved
6. Results need to be carefully analyzed.

Alpha Testing

Alpha Testing - Alpha testing is done by internal developer and QA teams, with its main goal being to ensure that the software is functional, reliable, and free of any defects or errors

- Prior to Beta
- Product stability still poor; more ad-hoc process

Beta Testing

Typically for products; More so for new product releases
Product rejections in market place is a huge risk.
Reasons –

- Implicit requirements not addressed
- Changed needs/perceptions after the initial specs
- Usability
- Competitive comparisons by users

Beta Testing

Process

1. Select & list representative customers
2. Work out a beta test plan
3. Initiate the product and support throughout
4. Carefully monitor / watch the progress and the feedbacks – good & bad both
5. Have a good response system to avoid frustration for customers
6. Analyze the whole feedback and plough it back for product improvement

Incentivized participation

Alpha Testing vs. Beta Testing

Alpha testing vs. Beta testing

- Alpha and beta tests are both types of acceptance tests.
- During alpha testing, data is not real and typically, the data set is very small in order to make debugging and root cause analysis easer.
- Beta test participants are potential customers who have agreed to test a possibly unstable application.
- The users create their own data sets and the test focus changes to usability and evaluating real-life performance with multiple users using their own hardware.

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 4

Non-Functional Testing

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

PES UNIVERSITY

List of Contents

Overview

Functionality Testing

- To evaluate the functional requirements of a system and focuses on the outputs generated in response to selected inputs and execution conditions
- Focus is on testing the product's behavior (both positive & negative) from a direct output point of view
- We have seen many aspects of it so far

Overview

Non-Functional test: Testing the abilities of the system.

- Reliability
- Scalability
- Performance
- Stress
- Interoperability
- Security
- Compatibility

Why Non-Functionality Tests?

- ➔ To find design faults and to help in fixing them.
- ➔ To find the limits of the product
- ➔ Max no. of concurrent access, min memory, max # of rows, max
- ➔ To get tunable parameters for the best performance
- ➔ Optimal combination of values
- ➔ To find out whether resource upgrades can improve performance? - ROI
- ➔ To find out whether the product can behave gracefully during stress and load conditions
- ➔ To ensure that the product can work without degrading for a long duration
- ➔ To compare with other products and previous version of the product under test.
- ➔ To avoid unintended side effects.

Common Characteristics of NFT

NF behavior

- Depends heavily on the deployment environment
- “multi-” in most of the control parameters

Users, login sessions, records, durations…..

- Environment setting is elaborate, expensive.
- Execution environment also needs to be carefully controlled
- Noting down the version number of each component is very important.

Quick Definitions

Scalability test
Testing that requires to find out the maximum capability of the product parameters.

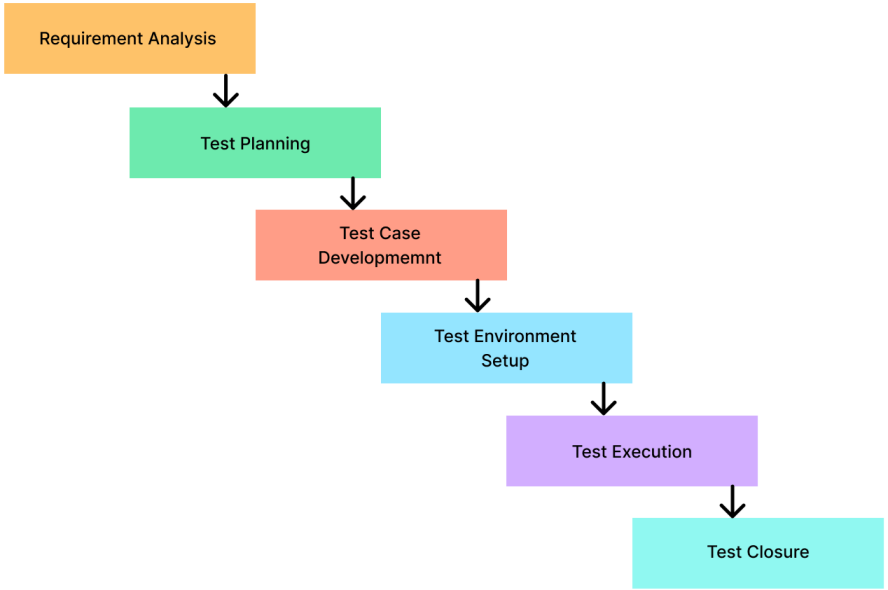
Performance test
Testing conducted to evaluate the time taken or response time of the product to perform its required functions under stated conditions in comparison with different versions of same product and competitive products.

Quick Definitions

Reliability test
Testing conducted to evaluate the ability of the product to perform it's required functions under stated conditions for a specified period of time or number of iterations.

Stress test
Testing conducted to evaluate a system beyond the limits of the specified requirements or environment resources (such as disk space, memory, processor utilization, network congestion) to ensure the product behavior is acceptable.

Test Phases


Requirement Analysis
Test Planning
Test Case Developmemnt
Test Environment Setup
Test Execution
Test Closure

Test Planning – Test Strategy Basis

- What are all the input's that can be used to design the test cases
  - Product Requirement Document
  - Customer Deployment Information.
  - Key NF requirements & priorities
- Industry / competitor products / current customer product behavior data for benchmarking.
- What automation can be used.
- Test execution in stages
- How much % of TCs need to/can be automated.

Test Planning - Scope

- The new features which can be attributed to non-functional quality factors
- Old features – design, code change, or side effects
- May skip unchanged and unaffected features

Test Planning - Estimations

- Effort estimation – More time
- Resource estimation – More H/W
- Defect estimation – Defects complex and costly.
- Reasons? - Difficulty in visualizing
- More, as compared to functional test runs.
- How much more etc, is matter of judgement.

NF Test Planning – Entry/Exit Criteria

When to start execution

- Product does not have basic issues
- Meets the minimum criteria

When to end execution

- Test results sufficient to make analysis / judgement
- Align to product release schedule

Entry/Exit Criteria - Examples

	Test Parameters	Entry Criteria	Exit Criteria
Scalability Test	Limits	Product should scale up to 1 million records	Product should scale up to 10 million records
Performance Test	1. Response 2. Throughput	query for 1000 records should be less than 3secs	query for 10000 records should be less than 3secs
Reliability Test	Failures / Iterations Failures / Test Duration	There should be less than 2% failures when queries are run on 1000 records for 24hrs	There should be less that 0.1% failures when queries are run on 1000 records for 48hrs
Stress Test	system when stressed beyond the limits	Product should be able to withstand when 25 clients logins happen simultaneously for 5 hrs	Product should be able to withstand when 100 clients logins happen simultaneously for 5 hrs

Test Planning – Defect Management

How is it different in NFT
Definition of defect is different!

- Defect logging (Same)
- Defect Reproduction – not easy
- Defect Analysis - not easy
- Defect Fixing – could start a series of activities.
- Defect Regression

Test Design – Typical TC Contains

1. Ensures all the non-functional and design requirements are implemented as specified in the documentation.
2. Inputs – No. of clients, Resources, No. of iterations, Test Configuration
3. Steps to execute, with some verification steps.
4. Tunable Parameters if any
5. Output (Pass/Failed Definition) : Time taken, resource utilization, Operations/Unit time, …
6. What data to be collected, at what intervals.
7. Data presentation format
8. The test case priority

Test Design – Test Scenarios

Based on the quality factors and test requirements different scenarios can be selected

With the basic setup
With best tuning
With upgrading resources
In typical customer setups
To compare with the competitor products
To compare with the older version

What is Scalability?

Ability of a system to handle increasing amounts of work without unacceptable level of performance (degradation)

Vertical scalability (Scale Up)

Add resources to singe node (e.g. CPU, memory)

Horizontal scalability (Scale Out)

Add more nodes (e.g. computers, ATMs)

Example, server cluster

Trade-offs

Mgmt complex, programming /deployment complexity, network latency

Test Design – Scalability Test

The test cases will focus on to test the maximum limits of the features, utilities and performing some basic operations

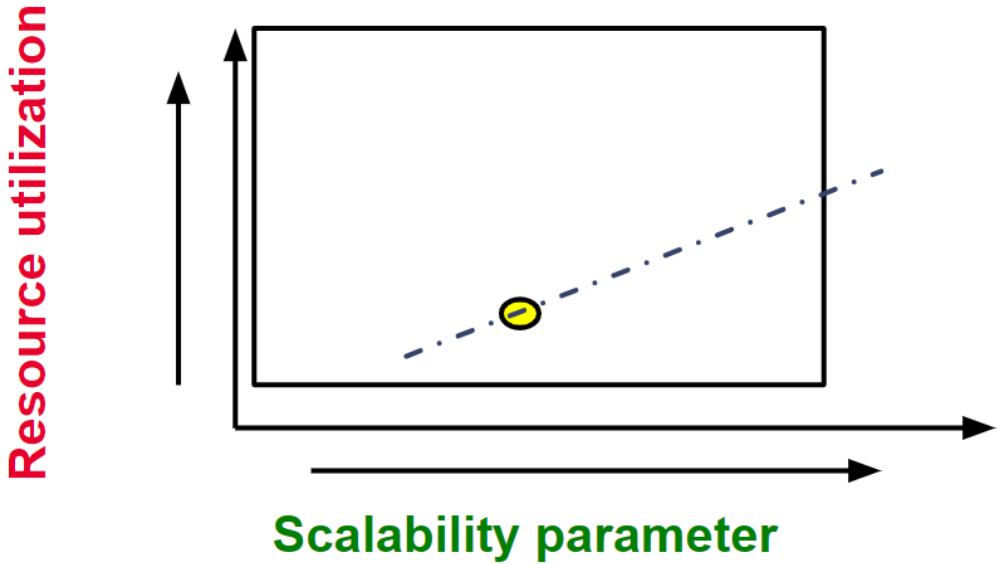
Few Examples:

Backup and restore of a DB with 1 GB records.

Add records to until the DB size grows beyond 2 GB.

Repair a DB of 2 million records.

Example of Scalability Test



Resource utilization is exponential beyond a point

Scalability Test– Outcomes

1. Based on a number of tests, documenting data and analysis:
    - Scalability increases N% for m times the resources – 50% increase if memory is doubled
    - Suitable , optimal configuration settings for a required scalability settings
1. This is an important part of application “sizing”
2. If required scalability not achieved, analysis and remediation done and retested.

Reliability

Probability of failure-free software operation for a specified period of time OR number of operations in a specified environment

- Defects injected because of modifications
  - Regression
- Reliable s/w development is difficult
  - Large s/w continue to have bugs even after 10-20 years of operation
  - Why?
- Testing all paths is “Impossible”
- New ways of program failures – Ex, security breach

Test Design – Reliability Test

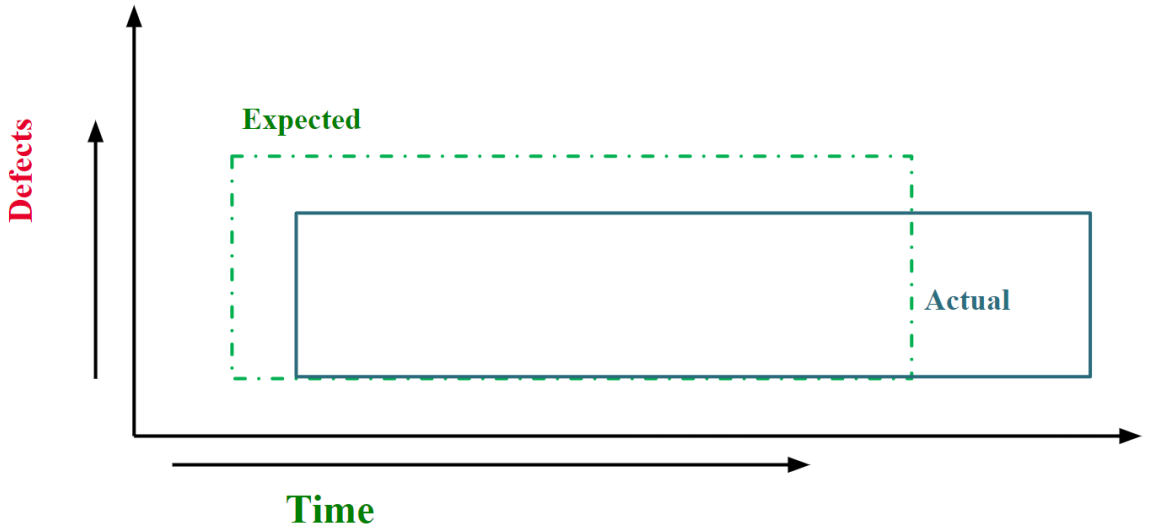
The test cases will focus on the product failures when the operations are **executed continuously for a given duration or iterations** – Long and many!

Examples:

Query for entries which are not available on a particular server, but available on the server which it is referring to continuously for 48 hrs

Focus is more on frequently used operations

Test Design – Reliability Test



Reliability Test – Outcomes

- # of failures in given run
- Interval of defect free operation - MTBF
- Some of the configuration parameter sensitivity – Which cause more/less unreliability
- Identifying causes of failures is hardest.
- Possible Failures reasons -?
  - Memory leaks
  - Deep defects due to uninitialized values
  - Weak error handling
  - Unstable environment
  - Unintended “side effects”

Test Design – Stress Test

To test the behavior of the system under very severe conditions – high load / low resource
Good system should show a graceful degradation in output and safe/acceptable behavior under extremes

Example:

Performing login, query, add, repair, backup etc operations randomly from 50 clients simultaneously at half the rated resource levels – say half memory / low speed processor…

Since stressed conditions are randomly applied over a period of time, this is similar to reliability tests

Stress Test - Techniques

- Run the application with very high load
- Run the app with very low resource levels
- Run multiple concurrent sessions
- Vary the loads/ resources randomly
- Tools to
  - create artificial scarcity of resources
  - Create artificial high loads

Stress Test - Outcomes

- As the resource / load ratio decreases, performance should go down gradually.
- In extreme conditions, should stop / pause gracefully
- Should recover symmetrically when stress conditions ease.

Stress Test – Failure Cases

1. System hang permanently.
2. Runtime crash – unexpectedly
3. Affect other programs in the system badly or verse the full system is made unstable.
4. Once stress limit reached, does not recover when stress levels are reduced

Test Design – Performance Test

The test cases would focus on getting response time and throughput for different operations, under defined environment and tracking resource consumption when resources are shared with other systems

Idea is to measure accurately response time and throughput under a defined environment and load and infer its acceptability

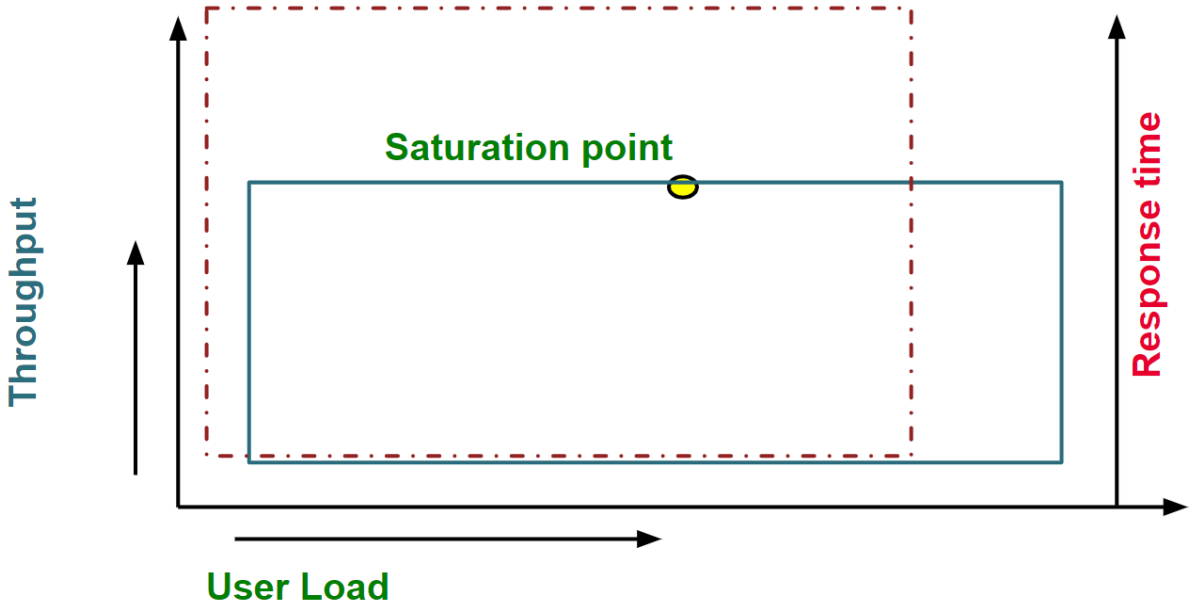
Example:

Performance for adding 1 million records

Performance Test – Multiple Needs

- To measure and improve the product performance
- To gauge performance against the competing or standard products
- To generate data for “product sizing” / capacity planning and product positioning in the market-place

Performance Test – Typical Outcome



Performance Test – Methodology

- Establish the performance requirements
- Design the performance test cases
- Automate
- Conduct testing and collect results
- Analyze results
- Tune performance
- Benchmark
- Establish needed configuration size.

Test Automation – Considerations

- Test automation is itself a software development activity
- Specialized tools and/or Shell script driven batch programs
- Input/Configuration parameters, which are not hard-coded
- Modularization and Reusability
- Selective and Random execution of test cases
- Reporting Data and test logs
- Handling abnormal test termination
- Tool should be maintainable and reliable

Test Setup

- **Non Functionality setups are usually huge**
- Different kinds of tests may need different setups
- Good idea to build the setup before execution.

Considerations for Test Setup

- Server – H/W, S/W
- Clients – Server Connectivity, H/W, S/W
- Network Topology
- Installation of debug, trace, record utilities on the test machines.
- Installation of pre-required software
- Uploading with the required data …

Test Execution - Objectives

In most of the cases the out come of non-functionality testing not only pass/fail definition, it's much more than that, i.e. data collection, data analysis and trend analysis.

Test objectives:

- Testing to improve the product quality factors by finding and helping infixing the defects
- Testing to gain confidence on the product quality factors
- Tunable parameters

Test Execution – Data Collection

Important part of NFT is data collection.
Mostly tool driven

Example

Time	CPU Utilization	Memory Utilization	Network Utilization (Number of packets sent)
01:00	77 %	17 MB	2.5 k
02:00	65 %	19 MB	2 k
03:00	67 %	19 MB	2 k
04:00	71 %	19 MB	2.2 k
05:00	77 %	19 MB	2.5 k
06:00	58 %	19 MB	1.6 k
07:00	63 %	19 MB	2 k

Test Execution – Outcomes

Observed during execution

- System crash / instability
- System not responding
- Repeated failures

Observed during Data/Trend Analysis

- Response times / throughput as the number of operations increase.
- Performance not correlation with resource upgradation
- Resource usage

Test Analysis - Sample Charts

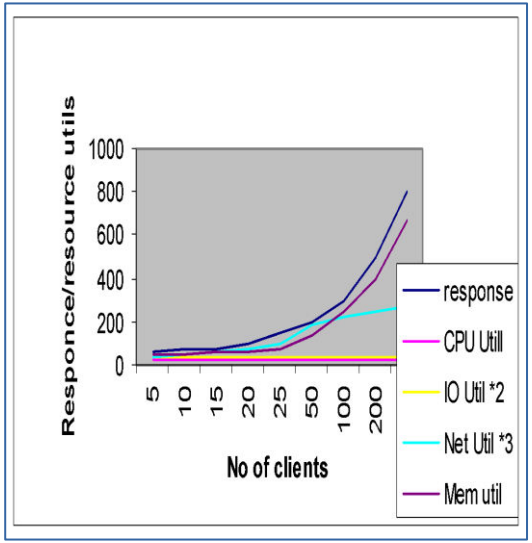
We can also draw the Throughput Chart and Response charts, combined with resource utilization of all the resources by multiplying and normalizing the data to find out

- What resource upgrades can yield better results?
- Whether there is any design issue?
- Whether certain kind of product behavior is acceptable in customer environment
- Different graphs provide different kind of data

Test Analysis - Scalability

Memory α Num clients ⇒
ROI is more when memory is upgraded as the number of clients increases.

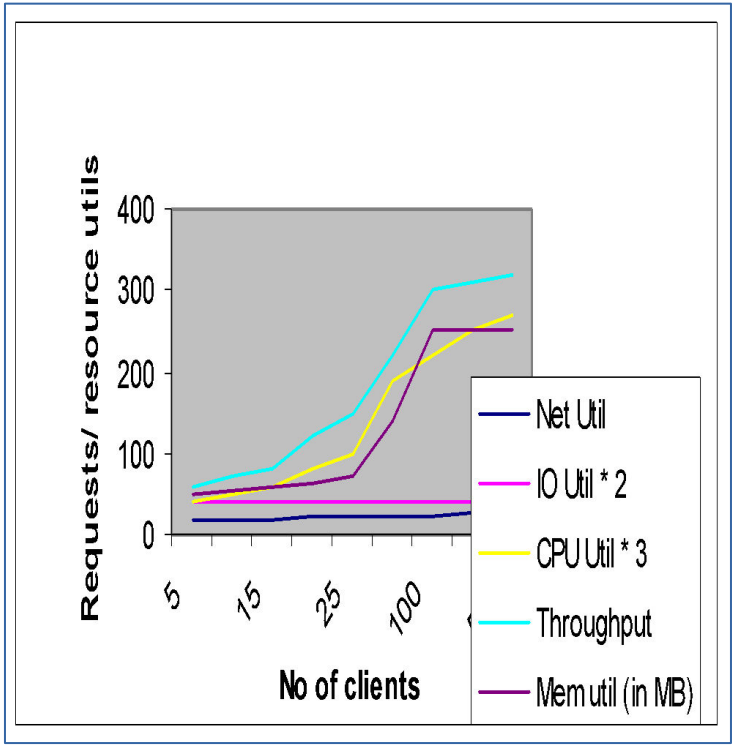
Network utilization α Num clients
⇒ This behavior of the product is not acceptable, if the customer environment is consisting of WAN Performance can be improved by increasing Bandwidth.



Test Analysis - Performance

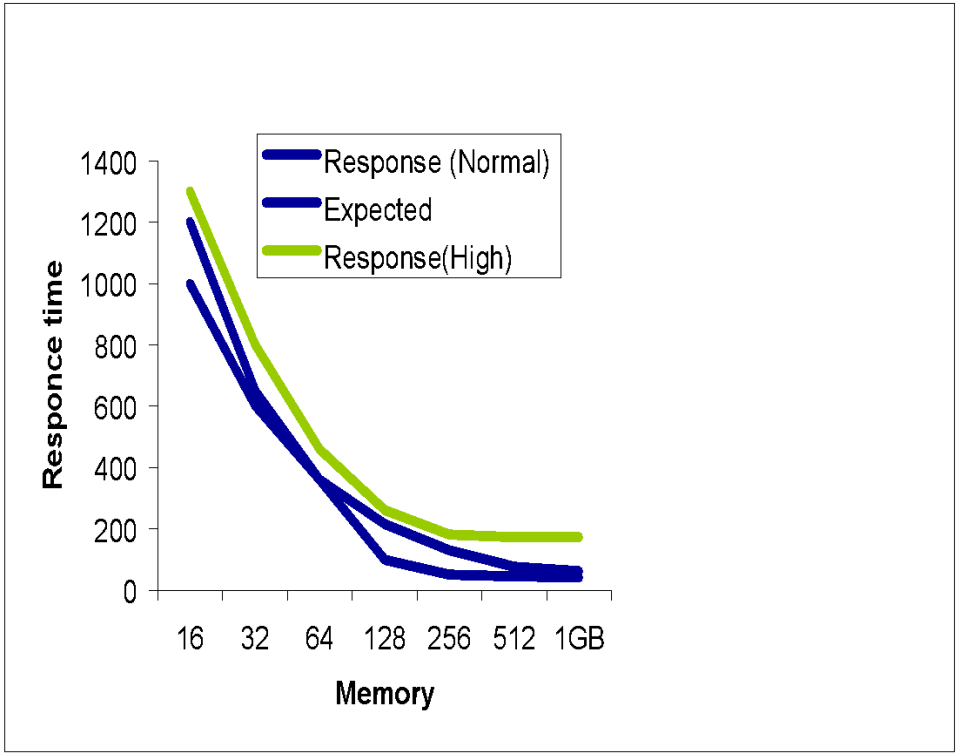
CPU and Memory α Num clients
Better performance by increasing these resources

Memory
After 100 clients, creation of process is failing or memory allocator in the S/W is failing
Indicates a defect or upgrade of memory after certain limit does not have any ROI



Test Analysis - ROI

Assuming the customers already have 128 MB RAM, The ROI from memory upgrade is short term, as the after 256MB there is no improvement in the performance.



NF Test Tools

1. LoadRunner (HP)
2. Jmeter (Apache)
3. PerformanceTester(Rational)
4. LoadUI (Smart Bear)
5. Silk Performer (Borland)

Security Testing

- Both static and dynamic
- Weak spots are called security vulnerabilities
- Many test tools to identify vulnerabilities at application level. Ex.,
  - Access control – application level
  - Direct usage of resources through low level code
  - SQL injection through input
  - buffer overflow
  - Usage of encryption
  - Sensitive info in non-secure channel(http)
  - API Interfaces
- OWASP Certifications / Standard

Few Other NF Test Types

1. Endurance testing
2. Load testing
3. Compatibility testing
4. Standards Compliance testing
5. Usability testing
6. Accessibility & Internationalization testing

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 4

Regression Testing

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

PES UNIVERSITY

List of Contents

- Regression Testing & Types
- Methodology
- Selecting Test Cases
- Classifying Test Cases
- Resetting Test Cases
- How to Conclude Results
- Popular Strategies
- Best Practices

What is Regression Testing

Regression testing is selective re-testing of the system with an objective to ensure that the bug fixes work and those bug fixes have not caused any unintended / undesirable side-effects in the system.

Not just defect fixes, even other modifications to call for regression testing.
Happens in fields other than software as well – Where a bit of complexity is involved.

Regression Testing – Types

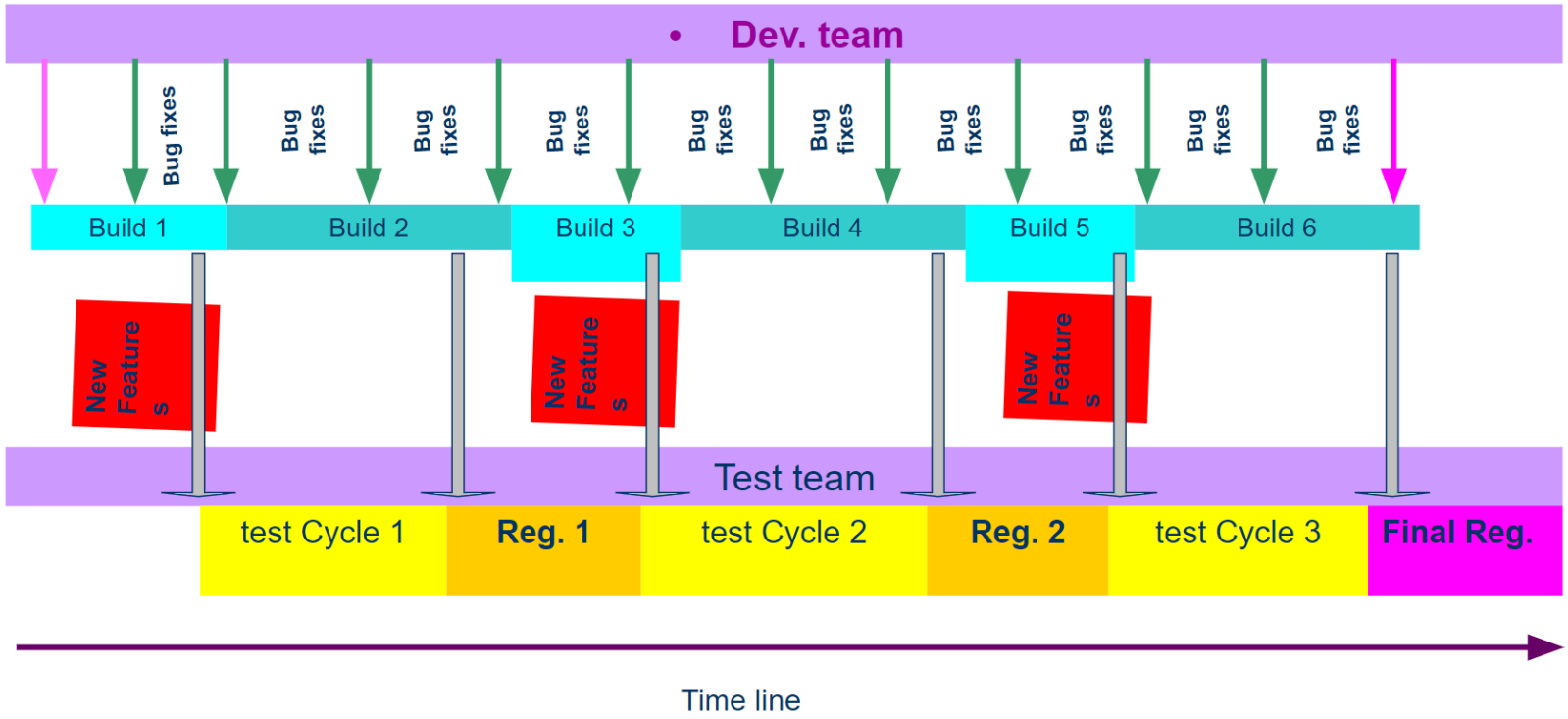
I. **Final regression testing**

- Unchanged build exercised for the minimum period of “cook time” (gold master build)
- To ensure that “the same build of the product that was tested reaches the customer”
- To have full confidence on product prior to release

II. **Regular regression testing**

- To validate the product builds between test cycles
- Used to get a comfort feeling on the bug fixes, and to carry on with next cycle of testing
- Also used for making intermediate releases (Beta,Alpha)

Regression Testing – Types



Regression Testing - Methodology

1. Criteria for selecting regression test cases
2. Performing smoke tests
3. Classifying test cases
4. Selecting test cases
5. Resetting test cases for regression testing & phases of testing
6. Conclude results
7. Popular strategies

Performing Initial Smoke Test

Smoke testing ensures that the basic functionality works and indicates that the build can be considered for further testing.

A ***defect*** is the cumulative effect of people, product, process and roles. Regression should take care of all these aspects.

What Is Needed for Selecting Test Cases?

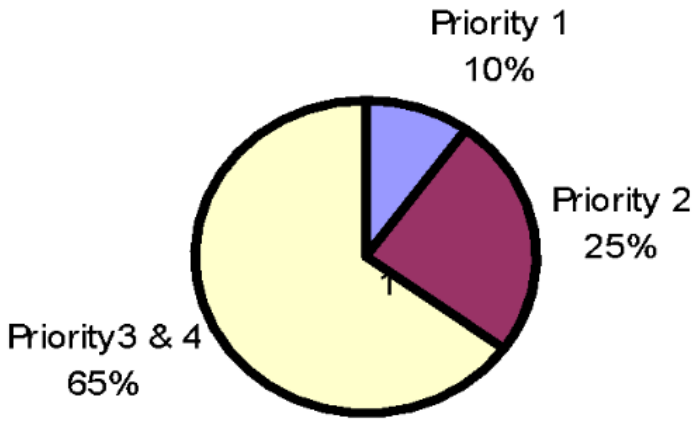
- Bug fixes and how they affect the system
- Area of frequent defects
- Area that has undergone many / recent code changes
- Area that is highly visible to the users
- Area that has more risks
- Core features of the product which are mandatory requirements of the customer

Points to Remember…

- Emphasis is more on the criticality of bug fixes than the criticality of the defect itself
- More positive test cases than negative test cases for final regression
- “Constant set” of regression test cases is rare

Classifying Test Cases

The order of test execution is priority 1, 2, 3 & 4. Priority helps in entry, exit criteria.



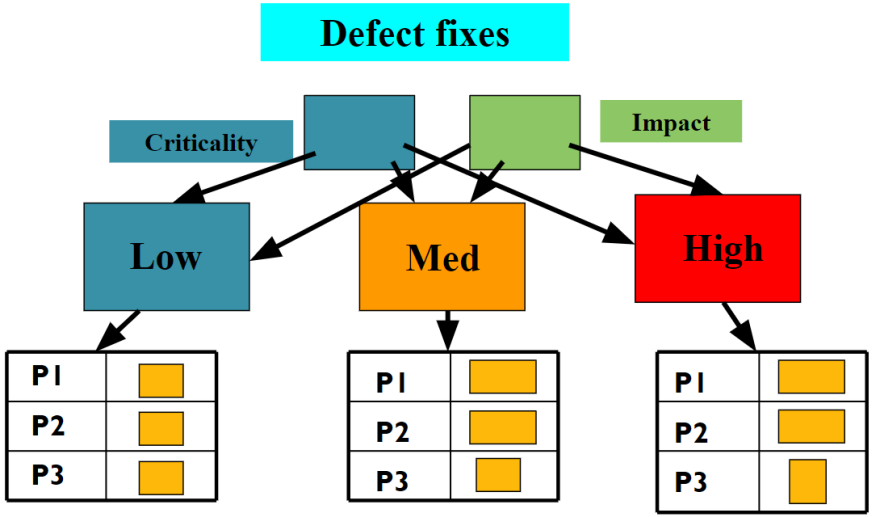
Classifying Test Cases

Priority 1 test cases	Priority 2 test cases	Priority 3 & 4 test cases
Sanity test cases, which check basic functionality Run before acceptance tests and for major change Delivers a very high project value	Uses the basic and normal setup Delivers high project value High probability of getting selected for regression	Uses extended set-up Moderate project value Some functional and all non-functional test cases fall into this category

Selecting Test Cases

Criteria

- Bug fixes work
- No side-effects



Resetting Test Cases

- The results of a test case can be guessed by looking at history.
- There is nothing wrong in communicating expected results before executing (but don't conclude).
- In many organizations not all types of testing and all test cases are repeated for each cycle (but the management generally wants overall statistics and 'gut feel')
- Re-setting a test case is nothing but setting a flag called NOTRUN or EXECUTE AGAIN and not getting biased with previous runs

Resetting Test Cases (Contd.)

It is done

- When there is a major change in the product
- When there is a change in the build procedure that affects the product
- In a large release cycle where some test cases have not been executed for a long time
- When you are in the final regression test cycle with a few selected test cases
- In a situation in which the expected results could be quite different from history

How to Conclude Results

Test case result (current build)	**Test case result (previous build)**	**Remarks**
FAIL	PASS	Regression failed and Apply RESET guidelines and proceed after getting new build
PASS	FAIL	Bug fixes are working and Continue your regression to find side effects
FAIL	FAIL	Bug fixes not working or not provided or Wrong selection
PASS with work around	FAIL	If you are satisfied with work around then result marked as PASS else FAIL (check with prog. mgmt)
PASS	PASS	This test case could have been included for finding side-effects or Wrong selection

Popular Strategies

1. **Regress all**. Rerun all priority 1 , 2 & 3 TCs. Time becomes the constraint and ROI is less.
2. **Priority-based regression**: Rerun priority 1 , 2 & 3 TCs based on time availability. Cut-off is based on time availability.
3. **Regress changes**: Compare code changes and and select test cases based on impact (grey box strategy).
4. **Random regression**: Select random test cases and execute. Tests can include both automated and not automated test cases
5. **Context-based dynamic regression**: Execute a few of the priority-1 TCs, based on context (e.g., find new defects, boundary value) and outcome, select additional related cases.

An effective regression strategy is the combination of all of the above, not any of them in isolation.

Some Guidelines

- Should not select more test cases that are bound to fail and have no or less relevance to the bug fixes.
- Select more positive test cases than negative test cases for the final regression test cycle as more of the latter may create some confusion and unexpected heat.
- The regression guidelines are equally applicable for cases in which major release of a product, have executed all test cycles and are planning a regression test cycle

Best Practices

1. Regression can be used for all types of testing and all phases of testing
2. Mapping defect numbers with test case result improves regression quality
3. Create and execute regression test bed daily.
4. Assign your best test engineers for regression.
5. Detect defects, protect your product from defects and defect fixes

Summary

Regression Testing

Types
Regular regression
Final regression

What?
Selective re-testing to ensure that:
- Defect fixes work
- No side-effects

Why?
Defects creep in due to changes

Defect fixes may cause existing functionality to fail

When?
When a set of defect fixes arrives after formal testing for those areas completed

Performed in all test phases

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 4

Agile & AdHoc Testing

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

PES UNIVERSITY

List of Contents

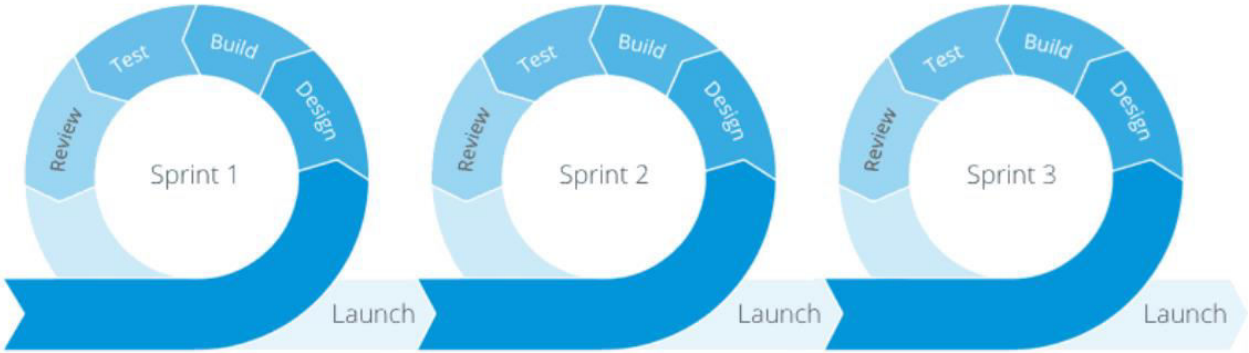
- Iterative Testing
- Agile Testing
- Methodology
- AdHoc Testing
- Defect Seeding
- Examples of AdHoc Testing

Iterative Testing

- Each of the requirements is at a different phase
- Testing needs to focus on the current requirement
- It should ensure that all requirements continue to work
- More re-testing effort

Agile Testing

Agile testing is software testing that follows the best practices of the Agile development framework. Agile development takes an incremental approach to development. Similarly, Agile testing includes an incremental approach to testing



Advantages of Agile Testing

1. Early Detection of Defects
2. Continuous Integration
3. Risk Reduction
4. Enhanced Product Quality
5. Cost-Efficiency

Agile Testing Methodology

1. Impact assessment
2. Planning
3. Daily stand-ups
4. Reviews

Defect Seeding

Defect seeding is a method of intentionally introducing defects into a product to check the rate of detection and residual defects.

It is a technique used in software development to determine the rate at which software development tests **detect errors** and the number of undetected errors in the system

AdHoc Testing

When a software testing performed **without proper planning and documentation**, it is said to be Adhoc Testing.

AdHoc Tests are done **after formal testing** is performed on the application. AdHoc methods are the least formal type of testing as it is **NOT a structured approach**. Hence, defects found using this method are hard to replicate as there are no test cases aligned for those scenarios.

AdHoc Testing Examples

When you want to …	The most effective *ad hoc* testing technique
Randomly test the product after all planned test cases are done	**Monkey Testing**
Capture the programmatic errors early by developers and testers	**Buddy Testing**
Test the new product / domain / technology	**Exploratory Testing**
Leverage experience of a senior tester as well as of a newcomer	**Pair Testing**
Deal with changing requirements	**Iterative Testing**
Make frequent releases with customer involvement	**Agile / Extreme testing**

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 4

Software Testing Tools

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

List of Contents

Software Testing Tools

Software Testing tools are the tools that are used for the **testing of software**. Unit testing and subsequent integration testing can be performed by software testing tools.

Types of Software Testing Tools:

1. **Static Testing Tools**

   Examples: Path Tests & Coverage Analyzers

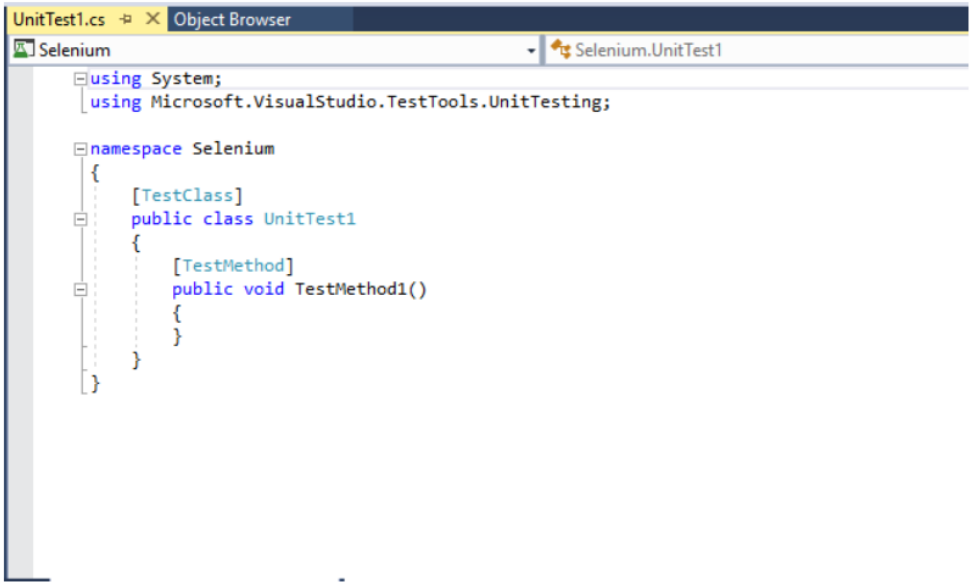
1. **Dynamic Testing Tools**

   Examples: Test Beds & Emulators

Selenium

Selenium is an open-source, automated testing tool used to test web applications across various browsers.

Selenium can only test web applications, unfortunately, so desktop and mobile apps can't be tested



Advantages & Disadvantages of Selenium

Advantages of Selenium

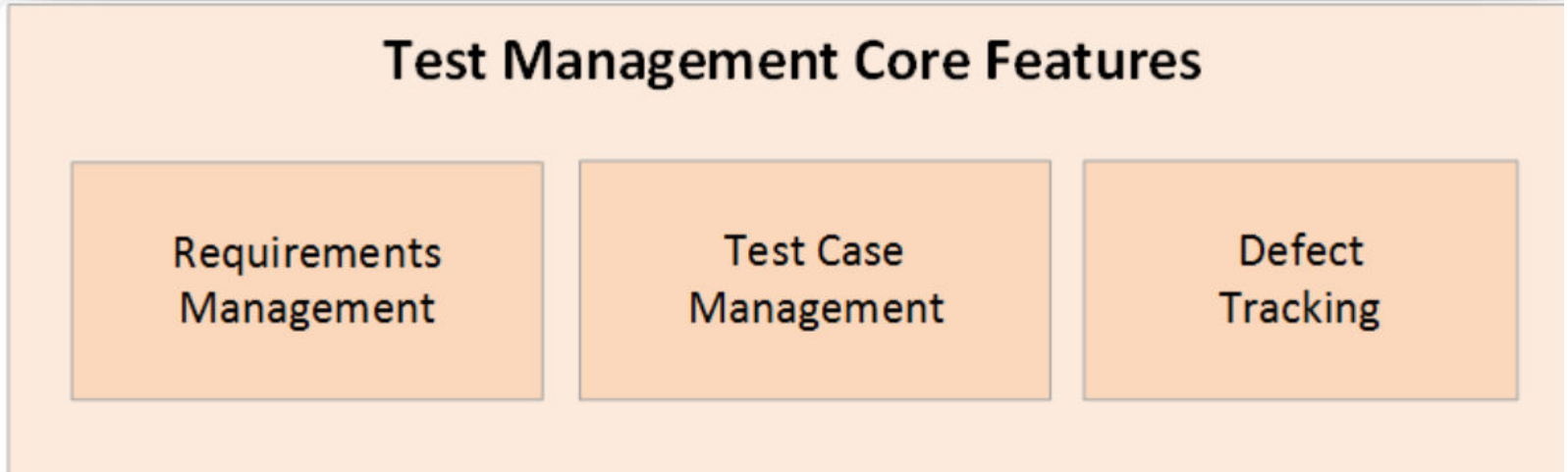
- Open Source
- Highly Extensible
- Run tests across different browsers
- Supports various operating systems
- Execute Tests in parallel

Disadvantages of Selenium

- Can only test web applications
- No built-in object repository
- Automates at a slower rate
- Cannot access elements outside the web application under test
- No official User support

Test Management Tools

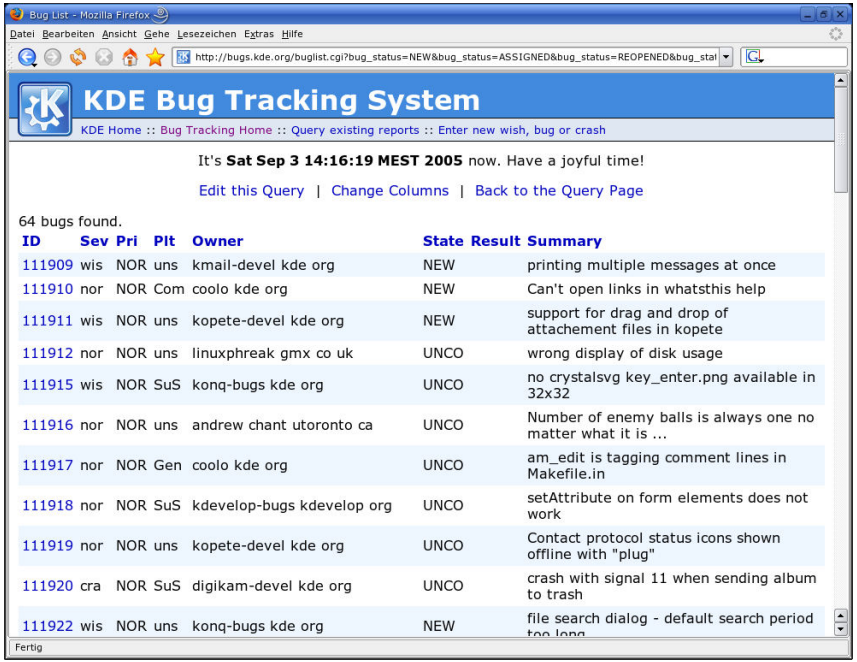
Test management tools are used to store information on how testing is to be done, plan testing activities and report the status of quality assurance activities.
Examples include Bugzilla & Jira



Bugzilla

Bugzilla is an open-source tool **used to track bugs and issues** of a project or a software. It helps the developers and other stakeholders to keep track of unresolved problems with the product.

It provides you a **very advanced search system** where you can create any type of search that you want such as time-based searches



Features of Bugzilla

Reference

Advantages & Disadvantages of Bugzilla

Advantages of Bugzilla

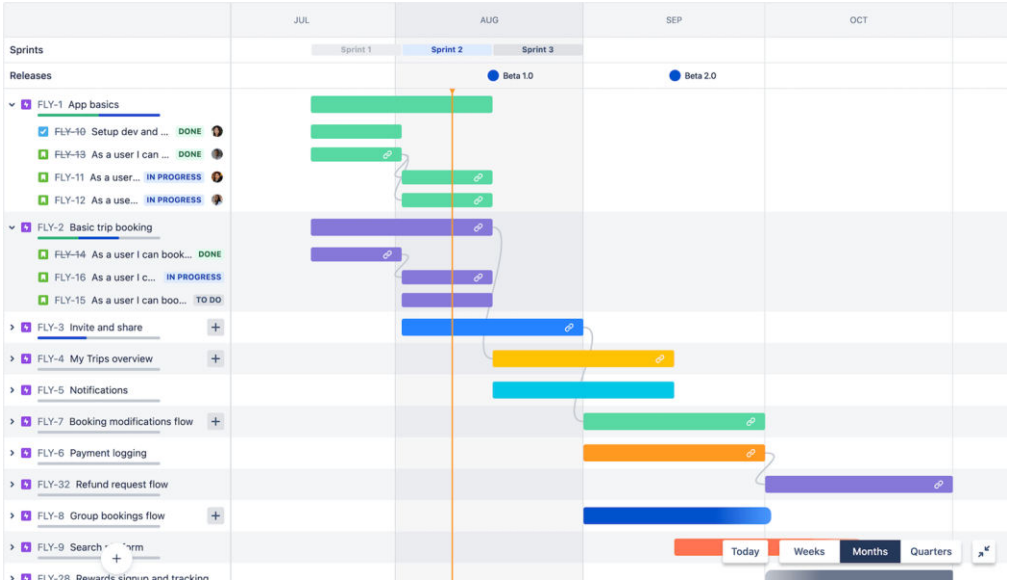
- It enhances the communication between the developing team and the testing team.
- Open Source Tool
- Provides advanced search capabilities

Disadvantages of Bugzilla

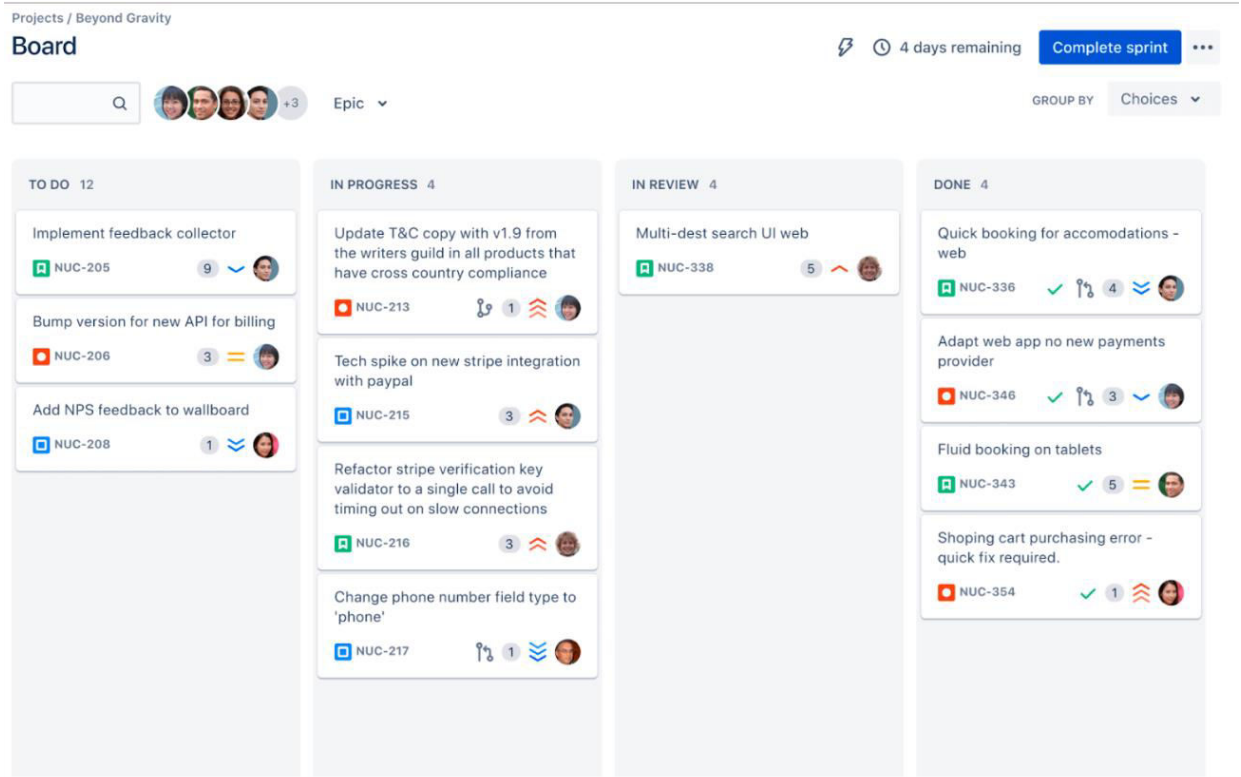
- No drag-and-drop reprioritization of bugs
- Poor user interface
- Does not support **Kanban Projects**
- Not recommended for Agile Project Development

Jira

Jira is a commercial software application developed by Atlassian for **bug tracking** and **agile project management.**



Jira


Projects / Beyond Gravity
Board
4 days remaining
Complete sprint
+3
Epic
GROUP BY
Choices
TO DO 12
Implement feedback collector
NUC-205
9
Bump version for new API for billing
NUC-206
3
Add NPS feedback to wallboard
NUC-208
1
IN PROGRESS 4
Update T&C copy with v1.9 from the writers guild in all products that have cross country compliance
NUC-213
1
Tech spike on new stripe integration with paypal
NUC-215
3
Refactor stripe verification key validator to a single call to avoid timing out on slow connections
NUC-216
3
Change phone number field type to 'phone'
NUC-217
1
IN REVIEW 4
Multi-dest search UI web
NUC-338
5
DONE 4
Quick booking for accomodations - web
NUC-336
4
Adapt web app no new payments provider
NUC-346
3
Fluid booking on tablets
NUC-343
5
Shoping cart purchasing error - quick fix required.
NUC-354
1

Some Jira Use-Cases

1. **Project Management**: JIRA provides a centralized platform for managing software development projects, with support for multiple projects and workflows.
2. **Task Management**: Teams can create, assign, and track tasks, bugs, and other types of issues.
3. **Agile Planning**: JIRA supports agile methodologies such as Scrum and Kanban.

Advantages & Disadvantages of Jira

Advantages of Jira

- Supports integration with other third party applications
- Suitable for different stakeholders like Software Testers, Developers and Project Managers
- Intuitive and simple User Interface

Disadvantages of Jira

- Commercial Tool - which can be used on purchasing the license
- Can be complex to use for new users
- Load times can be slow for large volumes of data

Bugzilla vs Jira

Aspect	Bugzilla	Jira
User Interface	Functional, somewhat dated	Modern and user-friendly
Integration and Ecosystem	Limited third-party integrations	Extensive ecosystem
Agile/Scrum Support	Limited support, may require plugins	Comprehensive support
Reporting and Analytics	Basic reporting	Robust reporting features
Cost	Open-source and free	Various pricing tiers (cloud and self-hosted)

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering