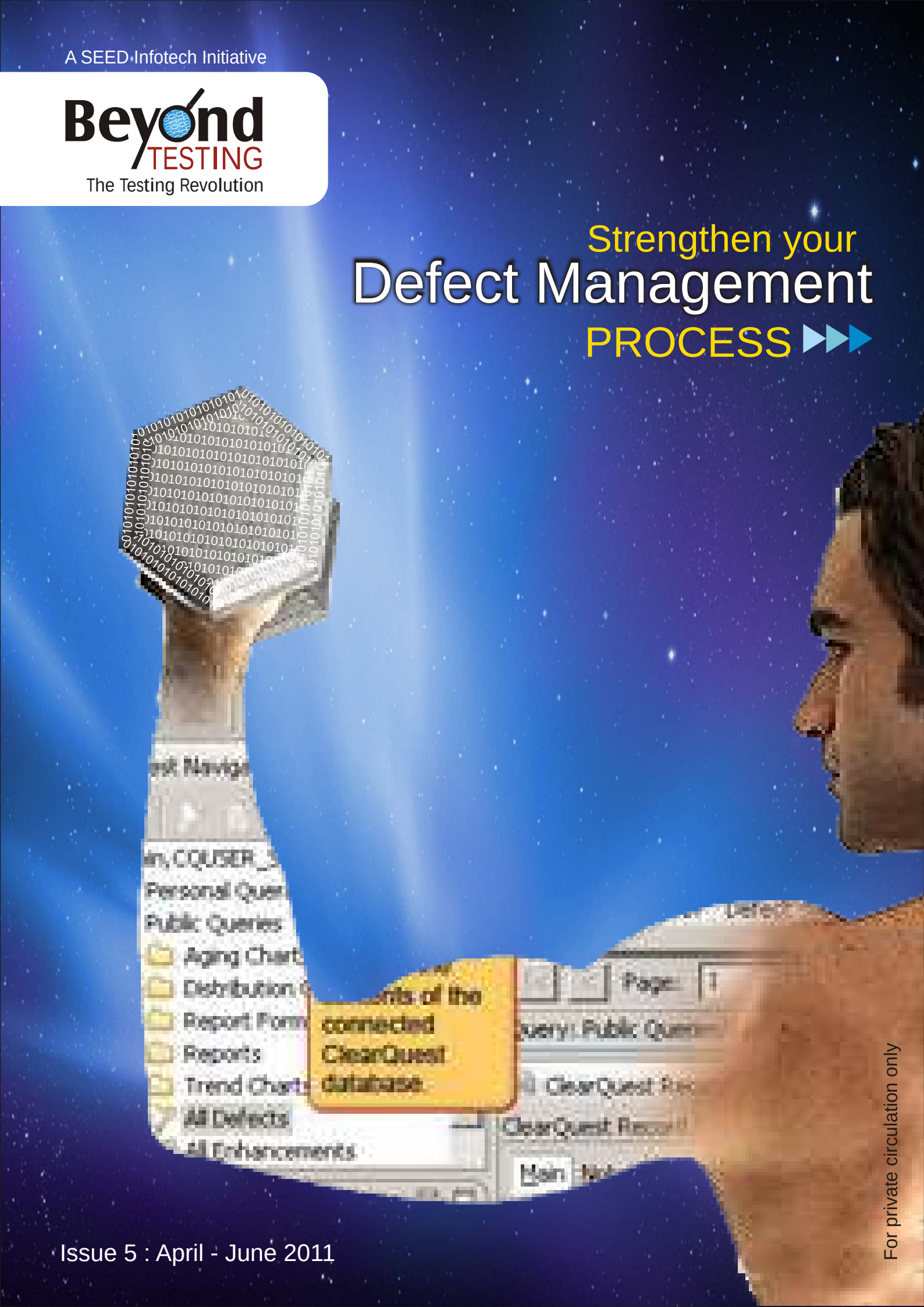
A SEED Infotech Initiative

Beyond TESTING

The Testing Revolution

Strengthen your Defect Management PROCESS



Issue 5 : April - June 2011

If bright career brings SMILES on your face, we are ready to help YOU.

2600+ Students placed in 12 months

Next could be you

Be a part of Software Testing Program

Our Programs:

- Functional Testing - Performance Testing - Security Testing - Usability Testing - Test Automation Tool

For registrations & more details contact :

SEED Infotech Ltd.: 'Panchasheel', 42/16, Erandawana, SEED Infotech Lane, Off Karve Road, Pune - 411004.

Tel.: 020-25467484, 25467484 | **Cell :** 98508 85179, 99229 33317

Email: sqa@seedinfotech.com | **Web.:** www.seedinfotech.com

seed™

beyond the obvious

For Campus Placements & Recruitment Solutions, Call : + 91 9822671630

SEED Infotech Ltd. Initiative

seed™
beyond the obvious

Beyond TESTING

Chief Editor
Jayesh !ngale

Board of Advisors
Narendra Barhate
Jayesh Ingale
Milind Limaye
Gireendra Kasmalkar
Rajesh Vartak
Rajesh Agrawal
Subodh Hawa

Production Team
Art & Graphic Designer : SEED Desgin Team
Printing & Publishing : Kedar Kakade

Feedback
feedback@seedinfotech.com

For Article
article@seedinfotech.com

Facebook
http://facebook.com/beyondtestingmag

Twitter
http://twitter.com/beyondtesting

Printed & Published by
Avani Publication

Cover Story

- Remove defects from the defect management process
- Effective defect reporting
- Handy Tips for Defect Prevention

Student Corner

Domain Corner

Column

28 pages including cover

Beyond TESTING

Contents

Chief Editor

It is sometimes amazing to observe how people in a software development team are carrying out different roles and performing range of activities. Wonder whether all of them are oriented towards one goal or multiple goals. Wonder whether the goals go hand in hand or whether they conflict with each other.

I asked couple of senior software professionals “When you are involved in product development, what is your primary or fundamental goal?”. Most of the respondents had one answer in common – Developing better Quality Product.

So I asked, what is going to prevent them from achieving this Goal. And pat came the reply – “Defects”. I asked “You have good testers, good developers, good managers, so should be an easy ride for you?”. To which I could hear, “Ya, ya… but No, no…”

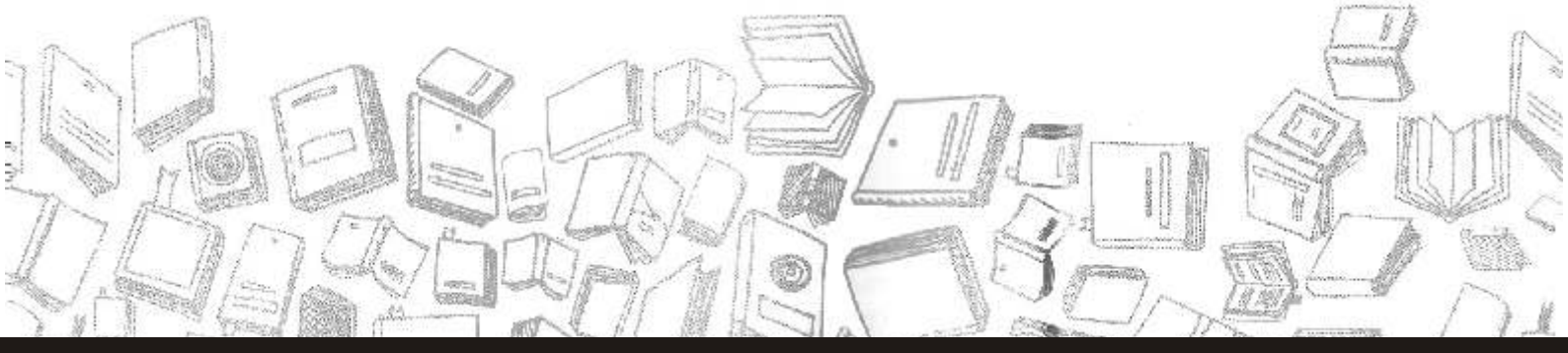
Hmm, so what’s missing is something which can manage these dangerous beasts “Defects”. This issue we focus on strengthening the defect management process. We have variety of articles starting from identifying defects in defect management process, to reporting and preventing defects.

Also I am glad to start another series viz QTP corner. This would certainly be useful to automation professionals who are using HP QuickTest Professional. If you recollect we had initiated a series last issue viz Domain Corner. This time we will have a look at VOIP testing. Apart from that we have columns written by experienced professionals and the one you should not miss is “Myths about Globalization Testing”

So enjoy reading this issue and let us know your feedback and suggestions to Jayesh.ingale@seedinfotech.com

Chief Editor

Jayesh !ngale



Uncovering myths about Globalization Testing

- Anuj Magazine

Prologue

In his landmark book- "The World is Flat", Thomas L Friedman has beautifully summarized the advent of World Globalization into 3 different eras-

Globalization 1.0:

Globalization 1.0 was primarily the globalization of countries that began with Vasco da Gama and Christopher Columbus started exploring the world. And that era, one could say, started in the 1400s and continued all the way - in its own way - right up to the invention of the steamship, the railroad and the telegraph, but ended with World War I. At the end of this era, the world gradually shrank from "Size Large" to "Size Medium".

Globalization 2.0:

This era was from World War II right up until year 2000, until Y2K-the era when the world continued to "shrink," when multinational companies increasingly went global for markets and labor, and when technological innovations continued to reduce transportation, communication, and production costs. Globalization 2.0 shrank the world from a “Size Medium” to a “Size Small” with the innovations in the field of telecommunication like Telegraph, Telephone, PC, Fiber Optics, and World Wide Web.

Globalization 3.0:

This period is something that really shrank the world further from "Size Small" to "Size Tiny". Globalization 3.0 is the intensification of everything that was invented in Globalization 2.0- the bandwidths, the fiber-optics, the PCs, and the software capabilities that connected them- but intensified to such a degree that it became a difference in kind.

We currently live in Globalization 3.0 era and in order to meet and further exceed the ruthless demands of this era, the Software products have evolved and grown in complexity. In addition to the Software being enhanced in features and Technical capabilities, one of the areas

that have evolved during this era is Software Globalization. As an example- Office 95 supported 27 different languages. Office 2007 supports more than four times as many languages and the list continues to grow as the worldwide market for software grows. With the world getting tinier by the day, the demand for high quality locale aware Software is high and continues to increase. The arena of Software Globalization has also seen a positive drift from Japanization to "Simship" in the past decade or so. Many new Globalization specific concepts have emerged during this era such as Unicode, Single Base Binary, Multilingual User Interface etc. and all this has led to Software Globalization testing become a separate discipline. While Software Globalization testing is still finding its feet as a separate discipline in many organizations, more mature organizations have embraced it and have reaped benefits successfully over the years. For anyone who is new to Globalization testing, the introduction to this discipline is often surrounded by pre-conceived notions and often resulting in misconceptions. This article primarily aims at uncovering a few major myths about Globalization testing. Here we go-

Myth# 1: Globalization (G11n) Testing is primarily about Testing the User Interface

The underlying Truth-
This is a myth because- to a majority of people, the translation of UI text and labels etc. are the only changes that get done when a product is globalized. There is no doubt that whenever an application gets translated to any other language, the UI changes are the most prominent in the product. All the text/labels that were originally displayed in the English language now get shown in the translated language. Though UI changes are the visibly significant changes, technically these are only a small part of the changes that get done as a part of Globalizing the Software. A few examples of activities involved in Globalizing the software include-

- Externalizing the UI.
- Design UI so that strings can expand.
- Ensuring that the product can support international character sets.
- Eliminating strings corruption.
- Ensuring that the application is locale aware.
- Support for bidirectional text.
- And many more...

A typical G11n testing cycle (especially after Localization Release To Test phase) might have more UI issues logged than the actual functional issues depending upon the quality of Globalization implementation, but that doesn't mean that majority of testing efforts gets spent to find UI issues only as there is a considerable testing effort spent to verify the other aspects of Globalization as well.

Myth# 2:

A person who doesn't know French cannot test the French version of the Software

The underlying Truth-
Other than the testing required for Language verification on the User Interface and the documents, every other aspect of Globalization testing can be done by a test engineer who is not well versed with the language under test. In fact, the Internationalization aspect of Globalization testing is usually done by Engineers who are not quite well versed with the language. Some facts-

- The User Interface of localized Software products is generally consistent with the English language User Interface. A tester first gets trained on the English version of the Software product and then switches to testing on a language specific version of Software. The tester gets acquainted with the entire User Interface map in English language and thus knows what options he/she is accessing on a language specific build.
- The language part of testing is best done by the native speakers of that language. There is a vast difference between native speakers and the person who has learnt a foreign language. It is generally very difficult to find a person who is technically a subject matter expert, an Expert in Software testing and a native language expert as well. If at all one manages to find one such person for one supported language, it may not be equally feasible to find such a person for all the supported languages. In such a scenario, the language part of Globalization testing (usually referred to as Localization testing) gets done by native language expert and the Functional part of Globalization testing (referred as Internationalization testing) is done by testing Experts.

Myth# 3:

A tester only needs to follow the test cases executed for English language in order to thoroughly test the Globalized applications

The underlying Truth-
In almost all the Software products, the International versions have the same features and functionalities as the English version of Software. Many people new to the Globalization testing tend to believe that the test cases meant for testing English version of the product can be used for testing the International versions of the Software. This is typically not correct. Let me try to put things in perspective here-

- It is true that for the sizable chunk of Globalization testing, the Base language (usually English) Functional test cases can be used. The prime reason of following this approach is that one of the objectives of Globalization testing is to ensure that the localized version of Software works as reliably on localized test environment as English language

product does on English language test environment.

- However, it is very costly and inefficient to run all the English language functional test cases for all the languages being supported. Suppose a Software product supports 4 languages, if one intends to run all the English language test cases for all the languages, then the overall effort would be 400% more- which is impractical. Usually, if a product is properly Internationalized- there would be a single code base catering to all the supported languages. If this is the case, there won't be much risk in executing the testing by striping the test cases across the languages, for a certain number of similar languages at least.
- It is equally untrue that Globalization testing constitutes "just" executing the English language test cases on localized languages. Apart of executing English based test cases on the localized test setup, there is a great deal of testing that needs to be done to test the International requirements of a particular language under test e.g. If you are testing German language Software, one needs to focus on German keyboard and testing using German characters, test for German date/time format, number formats, Sorting order, printer settings and a whole list of related things. Usually one ends up testing a whole lot of additional things.

Myth# 4: If a test case works fine in French language, it will work fine in German language as well

The underlying Truth-
This one is little tricky to explain but it is definitely a myth. Here are a few points around this-

- Lets consider a case of application being globalized for the first time. And assume that the application is going to support multiple languages. There are several factors that need to be kept in mind as the testing scope is decided-
  - First is to check if there are any changes in application binaries between the languages or all the languages use the same binary (Single base binary). If the binaries are the same across all the languages, then the next step is to know what changes have been done to the product from the Internationalization perspective across all the languages. Changes done to the product while creating

builds needs to be taken into account. Based on any of these changes, the results may differ across the languages.
  - If the application is being Internationalized for the first time, then the testing should be as thorough as possible as chances for mistakes are high.
- On the other hand, applications that have been through multiple Internationalization releases i.e. already support multiple languages, are generally going to be more stable and the variations of test results across the languages would majorly depend upon the changes that has gone into the Software between previous releases and the current one. In order to perform the Risk based testing across the different languages that are supported, one thought line that is usually applied is that all the Single byte languages such as French, Spanish, and German usually tend to behave the same way and the testing can be equally distributed across all these languages. Special care is usually taken to handle the testing of multi-byte languages such as Japanese, Simplified Chinese, Traditional Chinese etc. mainly because of the different character-sets that these languages deal with.

Myth# 5:

Globalization testing doesn't require the same test setup as is required to do the base language testing. Globalization testing can be done with a minimum test setup.

The underlying fact-
This statement may be typically true if we are taking about System II or System III testing setup but as far as the System I test setup is concerned (including OS, other third party software, hardware etc.) Globalization testing usually requires the same setup as System I English testing, though it will require a fully localized setup e.g. German Win 2003 server, German Exchange server, German keyboard etc.
Remember- one of the basic purposes of Globalization testing is to ensure that the International version of the application on the respective language test setup works the same as English language version would work on English test setup.

Remove defects from the defect management process

- Pradnya Paithankar

Introduction

For a tester, finding a defect in application is very crucial task. Because it's not only one of the most important deliverables in testing life cycle but it also reflects tester's performance and expertise to a large extent. Making this defect acceptable to developers is the further step which confirms tester's ability to understand functionality and user expectation. Nil or minimal defect leakage at end user's end builds confidence of customer in the testing process effectiveness.

However, there are many factors that impact this entire process adversely and it leads to

- Endless communication between developer and tester to confirm defect occurrence
- Defect rejection from developer
- Cancellation of defect due to non-reproducible behavior
- Duplication of defects
- No detection of other defects introduced during earlier defect fixing
- Repetition of the similar types of defects detected at customer end

How to avoid all these situations is the crucial challenge for every tester. Let us look at some important aspects of the defect management process to take care of these issues.

The idea is to remove defects from the defect and the defect management process.

What causes defects in defect management process?

Typically, the defect management process would involve following activities. There is a possibility of a "Defect" in any of these activities.

Let's take an example.

- You are testing a school management system and one of the modules is related to displaying student's marks. Along with subject wise mark list, the screen also displays the minimum and maximum marks obtained by the student in that particular exam.
- During testing for one of the test cases, you find that the application is not showing minimum marks fields correctly. You log a defect "Application not showing correct value for minimum marks".
- Developer makes it "not a defect" showing that at his end he could see that data is displayed correctly. You again check at your end and tell developer that for student XYZ, you got this error.
- Developer rechecks the issue and fixes the defect.
- You validate closure of this defect in next cycle checking with data corresponding to other student which you created in cycle 2 and close the defect.
- After application goes live, customer finds a defect that application is not showing minimum and maximum values correctly.

So what went wrong here? You tested, logged a defect, got is corrected and still customer finds the issues in the application.

Here are different reasons what could have gone wrong.

- You logged a defect with only one instance of testing and with one set of test data.
- You did not analyze what was the difference between the student record that developer used for the first time (claiming "Not a defect") and student XYZ
- You validated defect with different set of test data
- You missed checking "Maximum marks" field showing correct data during regression testing

Here are some ways how we can avoid such errors

Follow structured test case failure analysis

- Explicitly understand what is working and what exactly is failing
- Recheck failure in multiple other possible ways like
  - Alternate path for functionality flow
  - More test data with different combinations
  - Other set up or configuration
- Pin point the exact failure and then convert it into the defect
- Recheck the dependent functionality to assess impact of failure

Eliminate issues in defect logging

- Enter precise and concise defect description
- Include steps to reproduce defects
- Add test case number for reference
- Add snapshots wherever useful
- Mention requirement which you are testing
- Mention test data used to test
- Mention if any specific setting is done for application or test environment when defect occurred

- See if the defect is duplication of earlier defect detected in the same cycle before adding new
- Mention severity based on its impact on application functionality and testing schedule

Ensure complete defect verification

- Check for the defect closure by repeating the same steps you mentioned in defect description
- Use same set up and same test data used in defect situation for verification as well
- Check if the same defect is not occurring by varying other parameters, test data or set up

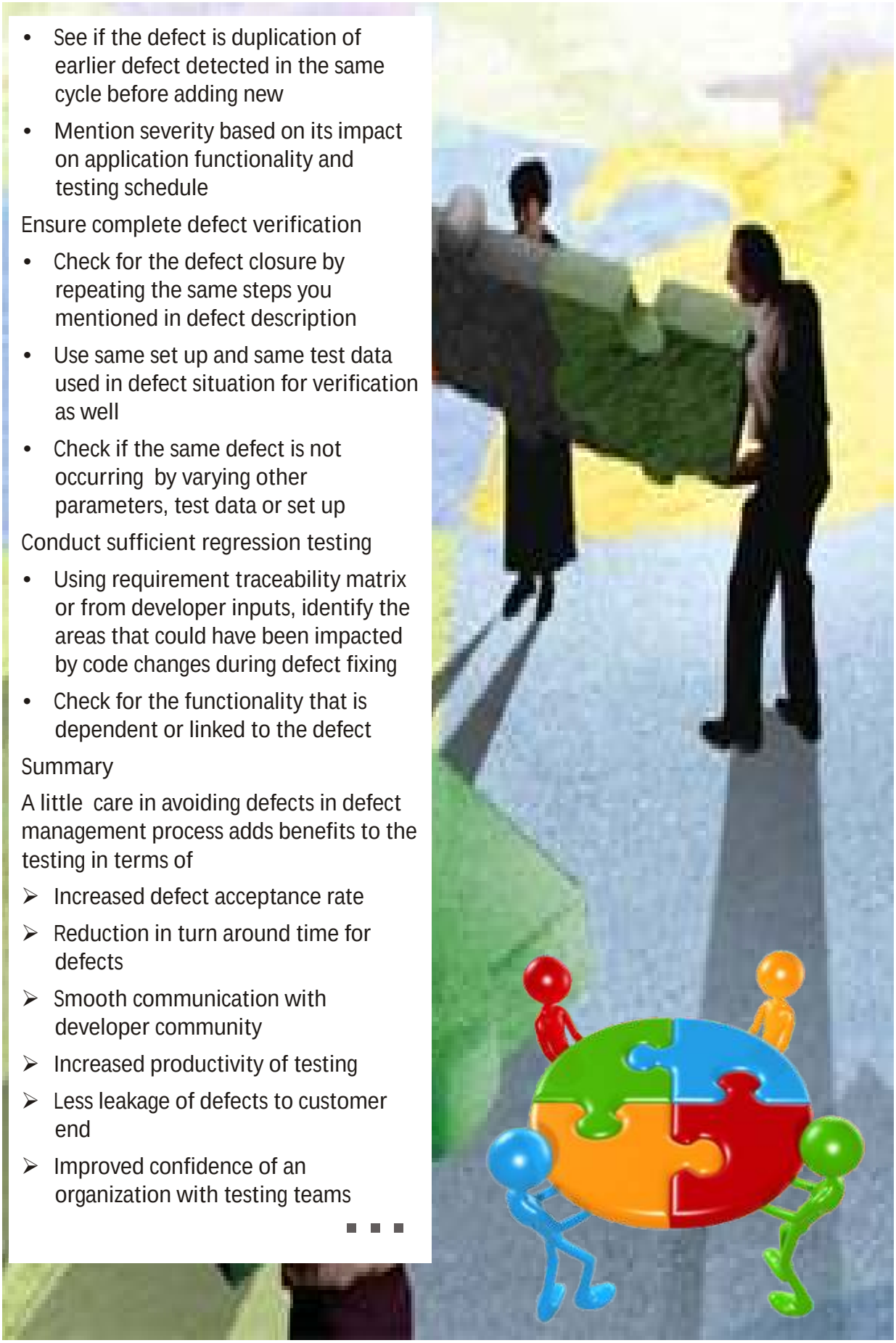
Conduct sufficient regression testing

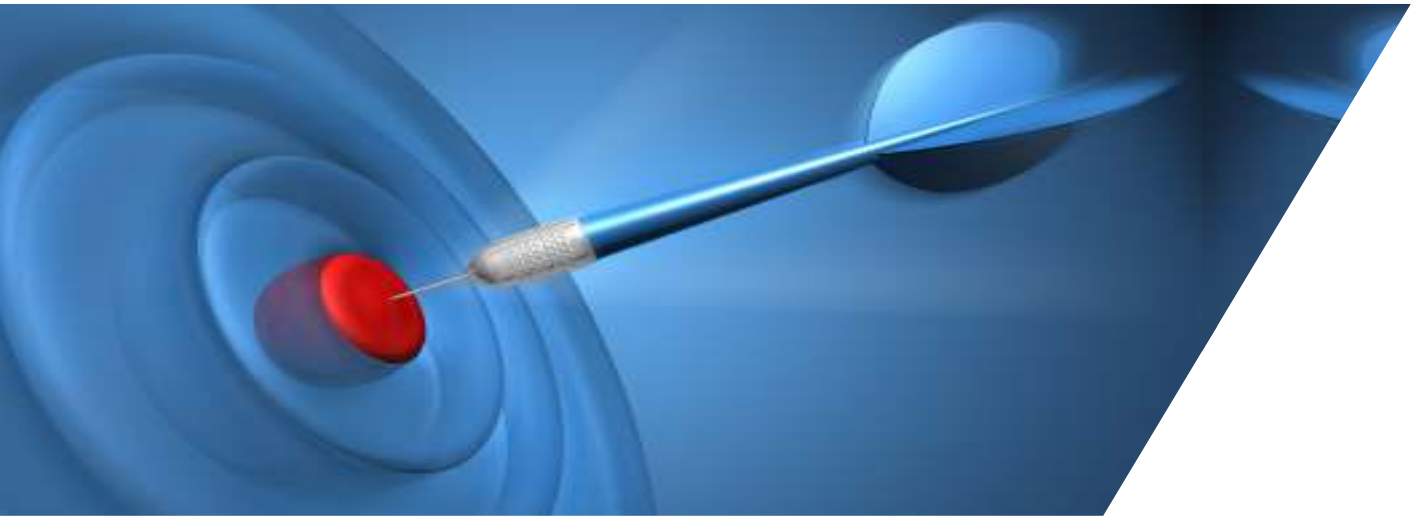
- Using requirement traceability matrix or from developer inputs, identify the areas that could have been impacted by code changes during defect fixing
- Check for the functionality that is dependent or linked to the defect

Summary

A little care in avoiding defects in defect management process adds benefits to the testing in terms of

- Increased defect acceptance rate
- Reduction in turn around time for defects
- Smooth communication with developer community
- Increased productivity of testing
- Less leakage of defects to customer end
- Improved confidence of an organization with testing teams





Effective defect reporting

- Vishal Shah

One of the aims of software testing is exposure to hidden defects. But only finding a defect is not enough, once you find any defect; you need communicate it through the whole team or to the concern person (for e.g. Business analysts, managers, developers, other team members, or client) in order to grab their attention towards it and get it resolved.

This process of communicating a defect or to notify a defect is called as defect reporting.

There should be some centralized place/database where reporter can file the defect and others can have access of it. To solve this purpose, many dedicated defect tracking tools are available; you can choose one of them as per your need.

Who all are involved in defect reporting process?

We can broadly categorized the roles, as Reporter, Reviewer, Developer and QA/Tester

Reporter is the one who reports the defect (can be a QA, can be a Client). After the defect is reported, Reviewer reviews it, reviewer can decide whether the defect is valid or not? If required reviewer can change severity and priority of the defect, even a reviewer can reject a defect. Once the defect is accepted it gets assigned to Developer. Developer fixes the issue and finally Tester confirms that the defect is fixed or not.

How to report a defect EFFECTIVELY?

Sometimes it happens that; though the defect is reported, developer needs to ask for further explanation to the reporter. One of the reasons is, reporter knows the defect but fails to communicate it properly or fails to provide sufficient information or sometime information provided is ambiguous.

Defects can be a complex or can be simple, but it is always very important to write a good defect report. As we saw that many persons are involved in defect life cycle, so a good defect report will be effective and easier for everyone to solve the problem. Ultimately it will help to save problem solving time.

Before reporting defects make sure that

- It is a defect and you can reproduce it
- It is not machine specific
- You are testing the correct version of the application
- Same defect has not already been reported

Reporting a Defect –

Different defect tracking tools might have different fields or different conventions. Attributes like Project, Version, Type, Severity, Priority, Summary, Description, Steps to reproduce etc. are mandatory and very important for reporting a defect are present in almost every defect tracking tool.

While reporting a defect –

- Select correct project name
- Select correct version
- Specify the Type whether it is a defect or enhancement request
- Choose right Severity and Priority based on the conventions followed by your organization.
  - Severity will express the impact of the defect on the application
  - Priority will decide the order in which the defect will get addressed and resolved. Based on this priority it will be decided which defect is more important and need to fix first.
- Summary – Summary can be a single liner description of the problem. Try to summarize the problem in a meaningful line so that just by seeing the summary one should get fair idea what the defect is about.
  - Summary should be clear and specific
  - Should not be ambiguous for e.g. "Accepting invalid data". This is example of bad summary. It does not talk about what data, on which page, when exactly it happens.
- Description – Elaborate the summary. Describe the problem in very simple and meaningful sentences. No Need to use very hi-fi, decorative words. Try to describe in the form of instructions (do this, do that, click here etc)
- It will be very helpful if you can provide detail steps to reproduce the defect, so that it will help developer to find and fix the problem.
- Always try to provide test data (where required) with the steps.
- If possible provide the root cause of the defect, it will save developers' time and efforts
- Also mention the expected result and actual result in the description or steps
- It always will be good practice to upload evidences (screenshot or error log file) with the defect. If possible highlight the exact problem in the screenshot
- Don't club or mix two defects, always file a separate defect report for each defect

Example – Lets us take a simple scenario for example

The scenario is, Tester is testing a user registration page and found that Birth date field is accepting future date.

Now let's see how the effective defect report will look like

Defect ID	00000XX	Date Submitted			1-Jul-11		
Project	Xyz	Reported By			Vishal Shah		
Version	x.xx.x	Status			New		
Type	Defect	Severity	High	Priority	High	Reproducibility	Always
Summary	User registration page is accepting future date as birth date						
Description	On user registration page, if we enter user's birth date as future date i.e. date greater than today, application accepts the date without showing any error message and allows the user to register. Expected - Application should show error message like "Birth date cannot be future date" and should not allow the user to register						
Steps to reproduce	1) Open the url www.exampletestssite.com 2) Click on the link "New user" 3) Fill all mandatory fields 4) Click on Calendar control against Birth date field 5) Select the future date (16-Aug-2011) 6) Click Register Button						

All these are very simple yet very important techniques to note and if you follow these techniques, I am sure it will make your defect report much effective.

Handy Tips for Defect Prevention

- Vishal Patil

In today's competitive market more emphasis is given on software quality and it has increased due to forces from several sectors of the IT industry. Defect Prevention is one of the most important activity of a software development which has a direct impact on controlling the cost of the project and the quality of the deliverables.

The new generation customer is smart and quality conscious. Customers, at times expect a complete report of defects. So it's very important for any organization to follow a robust defect prevention and detection process/ methods.

As we all know, testing starts from the requirement phase itself, but in this article I am going to focus on some of the key defect preventive measures which can be used during implementation phase

Defect Prevention: Defect prevention means avoiding defects before and during coding, instead of finding and fixing them afterwards.

Few suggestions on what could be done to prevent defects during software development process:

Standards: Agree on and commit to a mandatory set of coding and design standards. Issues to be covered: Design, C++ usage, documentation, complexity, maintainability, interface design, error-handling, coding in general, use of standard solutions for standard problems.

Code Reviews: Establish regular code peer reviews. Use the codes you work upon. Use coding standards as reference and/or discuss what is really important to you. Change partners often and across domain-borders.

Personal Favorite Defect List: Keep a private list of your "favorites" bug causes and standard violations. Use that for continuous improvement.

Expand Developer Tests: When coding, test your code as if there would not be any QA later and all user's satisfaction would depend on you alone.

Use Unit Tests: Write unit tests and run

them regularly. All unit tests have to pass.

Smallest Possible Steps: Adopt a habit of developing features in the smallest possible steps like, the cycle of develop -> test -> develop -> test ... should be as short as possible, usually much shorter than a day.

Warning Free Code: The compiler is your first and free check against uncertain code. Program such that there are no compiler warnings.

Dynamic Code Checks: Use tools to dynamically check your code against memory corruption.

Static Code Checks: Find and use tools that can statically check your code for correctness and maintainability. Currently most useful tools for this are not yet free/open-source but this may change.

Education: Educate yourself regularly. E.g. the knowledge about writing good C++ has changed enormously within the last 10 years.

Make Quality a Goal: Make the quality of your code just as important as the functionality. Indeed the quality is part of the functionality.

Plan for Quality: High quality coding needs more time than hacking. Plan for that. You get the time back by less bugs and easier code maintenance.

Analyze Bugs: Find out how bugs came into life, explore the reasons (inside and outside the code itself), and change what caused the defects.

Refactor: If code is difficult to understand or a continuous source of bugs, refactor it, until it is easy to maintain.

There are two measures that have a strong influence on the outcomes of software projects:

1) Defect potentials

2) Defect removal efficiency.

1) Defect potentials: The term “defect potentials” refers to the total quantity of bugs or defects that will be found in five software artifacts: requirements, design, code, documents, and “bad fixes” or secondary defects.

2) Defect removal efficiency: The term defect removal efficiency refers to the percentage of total defects found and removed before software applications are delivered to customers.

The Defect Removable Efficiency (DRE) is the percentage of defects that have been removed during an activity, calculated with the following equation:

DRE = (Number of Defects Removed / Number of Defects at Start of Process) * 100

The DRE can also be calculated for each software development activity and plotted on a bar graph to show the relative defect removal efficiencies for each activity. Or, the DRE may be computed for a specific task or technique (e.g. design inspection, code walkthrough, unit test, 6 month operation)

Example: We can also calculate DRE as:

DRE = A / (A+B)

Where A = Defects by raised by testing team and B = Defects raised by customer

A DRE of 0.8 or less is considered as a good score

Conclusion:

Defect prevention activity involves

1) Understand the mechanisms for defect detection and prevention.

2) Know how to collect, categorize and use defect information.

3) Find where to apply lessons learned.

4) Root cause analysis

5) Apply the defect prevention process.

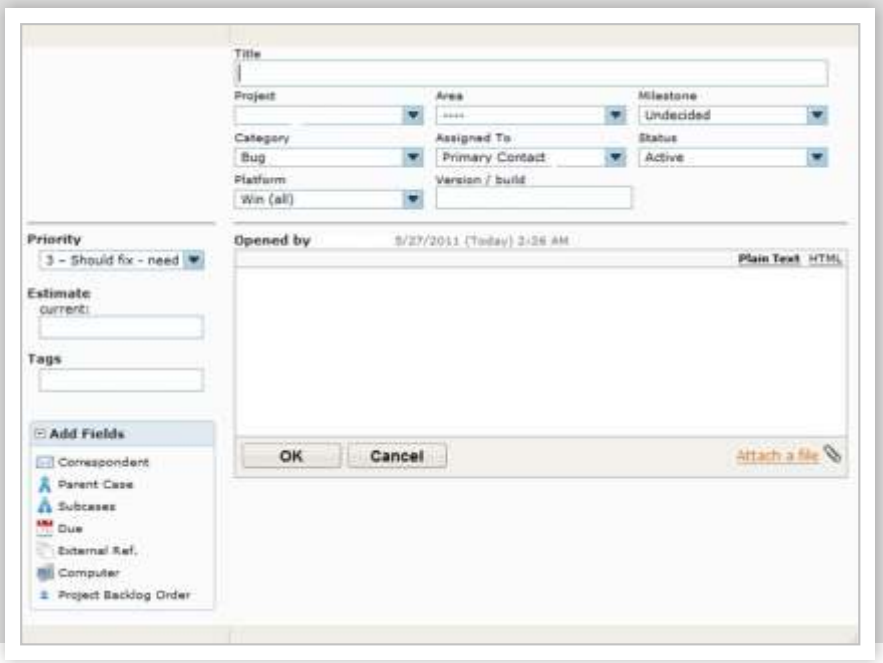
Defect Reporting for Dummies

- Shishir Laad

"Defect" or "bug"are important words in a software tester's life. In brief defect or bug is the deviation between actual and expected result, but only finding defect is not so important. As a software tester it's important to find defect as soon as possible and ensure that our bug should be fixed. Bigger the project and larger the number of defects present in the application under test, the more is the need for better management of those bugs.

Most organizations uses defect management tools which help to manage bugs and simplify defect management process. There are several defect management tools like bugzilla, assembla, fogbugz and so on, which are used currently in organizations. Most of the defect management tools have similar formats for reporting defects.

Following is the Snap shot of a bug management tool showing the fields while logging any bug and short description of those fields.



The following information will help developer to reproduce the defect and fix it.

Title: - This field is to give title of the bug. QA should give the title as such that reader will understand the bug by title and will not proceed to steps written below.

Project: - As the name of fields indicates, here we specify the project name on which we are working.

Area: - It defines the area of the bug that whether bug is of UI level or functional level etc.

Milestone: - Milestone field shows the name of next build or patch in which the bug will be fixed and QA can retest that.

Category: - Category defines that whether the logged issue is a defect or an enhancement which is depends upon the specification of project.

Assigned to: - The name of developer or person who will take the responsibility to fix that bug.

Status: - This field shows the current status of the bug, whether the bug is active or fixed. The possible status for a bug when it is fixed can be Resolved (fixed), Resolved (By Design), Resolved (Won't fix) depending upon the resolution of the issue.

Platform: - Platform defines the system configuration on which bug is found i.e. if bug is found on Mac machines then Mac will be selected and if bug is found on windows machines then windows platform will be selected. The information would be incomplete if version of operating system is not specified.

Version: - It defines the version of patch or build in which the defect has found.

Opened by: - In this field, QA name who is the owner of the issue should be specified.

Priority: - Priority represents the urgency of fixing a bug, it defines that which bug need to be fix first. Developer in general should fix all high priority bugs first.

Severity: - Severity represents "how bad a bug is."It defines the impact of the bug on other functionalities and application, For example, a bug that causes the program to crash would be considered high severity, while a minor spelling error might be assigned low severity.

Last but not the least; we need to write steps so that developer can reproduce the defect. One can also attach the snap shot or any image, video and log files of bug to support the logged defect.

Once the tester submits the defect report in the above format, it is now the developer's/owner responsibility to resolve it. The tester would then verify that the defect has really been fixed. If the tester finds the resolution of the defect correct, then defect is closed, else it is re-assigned to developer/owner for correct resolution. This process continues till the defect is resolved successfully. That's when we conclude that our defect report has really helped resolve a problem and has reduced customer frustration.

Why QuickTest Professional?

- Manjusha Kale

In the functional Test Automation Life Cycle, “Selection and Acquisition of a tool” is a very important phase. In this phase, several factors need to be considered for evaluation. Among which are Application Technology support, Capability of a tool, Ease of Automation, Script Maintenance and Cost of Acquisition. In this very first article of QTP corner, I would like to start the series with assessment of QTP on these Factors.

Technology Support

It is imperative for a functional testing tool to recognize the objects of AUT(Application Under Test). The latest version of QTP i.e. QTP 11.0 provides support to a wide range of technologies. These are; Web, Java, .NET, WPF, SAP, Oracle, Siebel, Peoplesoft, Delphi, Powerbuilder, Flex, Terminal Emulator, Visual Age, Stingray and Silverlight. The add-ins for respective technology helps QTP to identify a particular object, for e.g. Java textbox or Visual Basic button. QTP also has good support for latest Web 2.0 toolkits: Dojo, GWT, YUI, ASP .NET Ajax.

QTP allows you to test on different environments like Windows 2003, Vista, XP and Virtualization Technologies etc. and browsers such as Firefox 3.5, Internet Explorer 6, 7, 8.

Features of QTP

1. QTP provides a Recording facility for performing Test Steps. During recording, it stores various controls/objects from Application with their properties in the Object Repository. These properties help QTP to identify the object from AUT during script run.
2. Checkpoint is a very powerful feature of QTP which does comparison between Expected and Actual Result. There are various types of Checkpoints available to user such as Standard Checkpoint, Text Checkpoint, Bitmap Checkpoint, Database Checkpoint, Accessibility Checkpoint and XML Checkpoint.
3. QTP allows Data-Driven testing i.e. execution of the same script with different sets of data. It has datatable

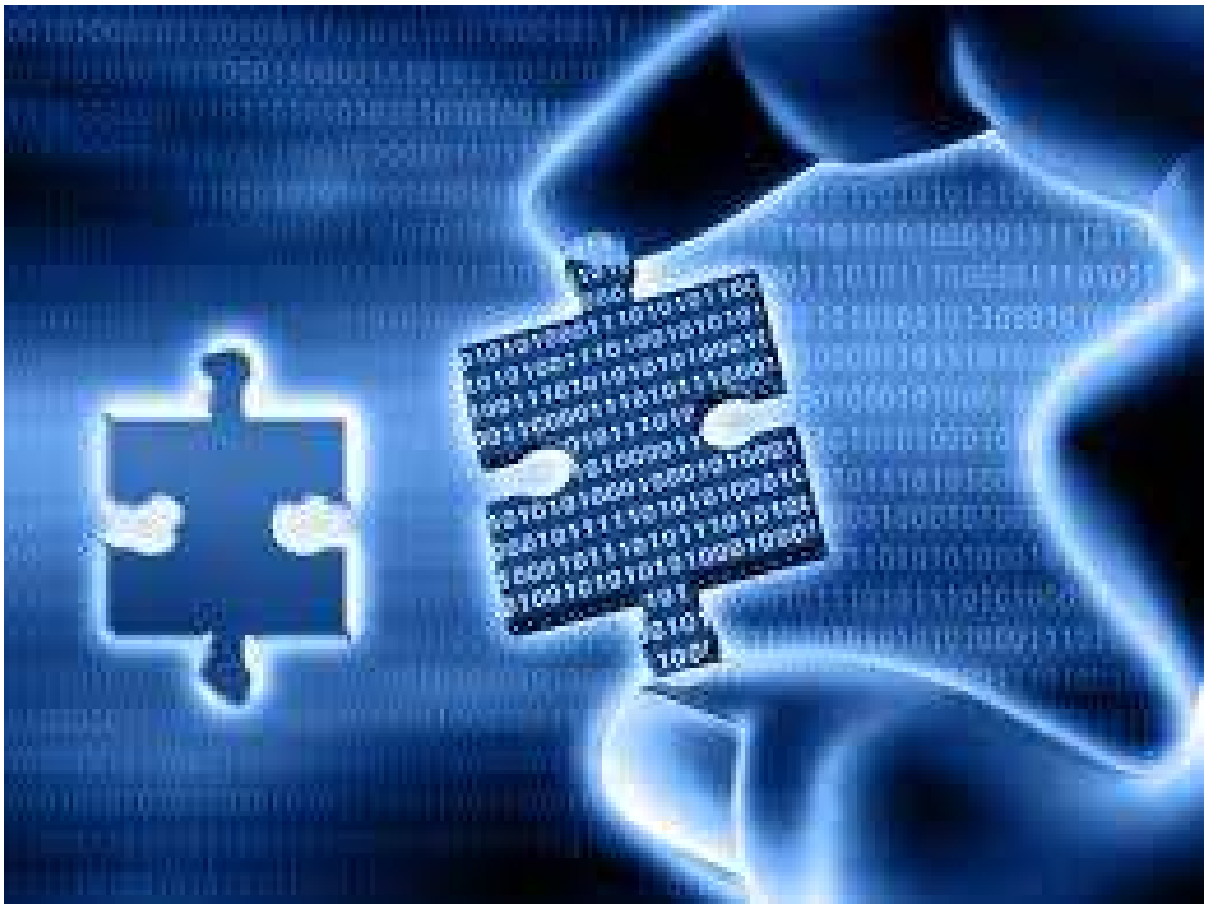
object which has several methods to iterate and select set of data. It is also possible to import and export data from and to Excel files.

4. QTP offers modularity to the script code through Actions. These Actions by default are Reusable.
5. User defined Functions can be created and re-used through Function Library.
6. QTP provides Debug facility to detect and isolate defects in a test script.
7. Its integration with QC allows selective execution of Test scripts.
8. It provides exception handling through Recovery Scenarios.
9. Objects which are not recognized by QTP but behave like some standard object can be defined as Virtual Objects and QTP can perform actions on them. They can be managed through Virtual Object Manager.
10. QTP supports COM. Hence it is possible to create objects of an application which follows COM. For e.g. you can create object of Excel, Word, Internet Explorer, Outlook etc. In fact user can even create QTP object which is known as Automation Object Model.
11. It can be integrated with Quality Center (QC), a Test Management Tool.

With this exhaustive feature list, you must have got a fair idea of its Capabilities. Now let's have a look at some additional facilities provided in QTP which make Test Script development enormously trouble-free.

Ease of Automation

1. Developing scripts in QTP is quite simple as it uses VB Script scripting language to describe test procedure.
2. There are two views provided by QTP,



Keyword view for the beginner and Expert view for those having knowledge of VB Script.

3. QTP IDE provides various Panes such as Information Pane, Missing Resources Pane, Available Keywords Pane and Active screen Pane these assist user during script development.

At times, Test Scripts also need to be changed due to some changes in the functionality. This comes under the Maintenance Phase of the Automation Life Cycle. However, this activity may result into lot of rework if there are no adequate features provided by the tool. QTP has taken care of these aspects also I am listing them below;

Maintenance of Scripts

1. QTP provides Maintenance Run mode. It is a wizard which guides you through the process of updating your steps and changes to object repository.
2. If you want to update Your Test Object Descriptions, Checkpoints, Output Values, or Active Screen Captures, QTP provide you with an option of Update Run Mode.
3. User can also ensure Reusability of objects by using Shared object repository. You can use the Object Repository Manager to create and maintain shared object repositories.
4. To increase maintainability, QTP allows parameterizing Test objects and data.

Cost of acquisition

I think I know exactly what is going on in your mind...the cost factor may prove to be overwhelming if you have budget constraints...don't give up just yet. I have some feasible solutions and options;

When you have a team of testers working in different countries and different time zones then go for Concurrent licenses instead of purchasing several Seat licenses.

Check whether you can manage with default add-ins so that purchase cost of additional add-ins can be avoided.

So let us conclude that QTP can be considered for automation of software applications and websites ranging from varied technologies. It has extensive capabilities of Test Automation. Its IDE provides easy interface for script development, execution and maintenance.

We would be exploring each of the above factors in detail in forth coming issues, till then strap your seat belt and be “Ready”.

VOIP Protocol Testing

- Shishir Laad

In this article I will be taking you through a brief tour on VOIP (Voice over Internet Protocol) protocol. I would then demonstrate how to perform testing of applications based on these protocols.

Every office and small business needs a method of running multiple communication programs such as conference calling, video conferencing, media distribution, instant messaging, and account management software. To perform these tasks, servers need the help of a gateway to bridge the gap between data streams and the Internet. Generally we use cell phones to call and send message but this technology allows us to communicate from one PC to another PC? This technology gives us the facility of audio call video call and instant messaging from one system to another PC or mobile. There are different components we need to use this technology. There are many applications like Skype, Bria, eyebeam and Ekiga and so on which are providing this facility over the world.

Following are some terms and info about this technology.

Protocol: A standard set of regulations and requirements that allow two electronic items to connect to and exchange information with one another. Protocols regulate data transmission among devices as well as within a network of linked devices through both error control and specifying which data compression method to use.

Protocols decides the method of error checking, how to compress data (if required), how the transmitting device signals that it has concluded sending data, and how the receiving device signals that it has completed receiving data.

Among the most common Internet protocols are FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), TCP/IP (Transfer Control Protocol/Internet Protocol), and SMTP (Simple Mail Transfer Protocol).

Voice Over Internet Protocol: Voice-over-IP (VoIP) enables users to carry voice traffic (for example, telephone calls and faxes) over an IP network.

There are 3 main causes for the evolution of the Voice over IP market:

- Low cost phone calls
- Add-on services and unified messaging
- Merging of data/voice infrastructures

A VoIP system consists of a number of different components: Gateway/Media Gateway, Gatekeeper, Call agent, Media Gateway Controller, Signaling Gateway and a Call manager.

The Gateway converts media provided in one type of network to the format required for another type of network. For example, a Gateway could terminate bearer channels from a switched circuit network (i.e., DS0s) and media streams from a packet network (e.g., RTP streams in an IP network). This gateway may be capable of processing audio, video and T.120 alone or in any combination, and is capable of full duplex media translations. The Gateway may also play audio/video messages and performs other IVR functions, or may perform media conferencing.

The coder-decoder compression schemes (CODECs) are enabled for both ends of the connection and the conversation proceeds using Real-Time Transport Protocol/User Datagram Protocol/Internet Protocol (RTP/UDP/IP) as the protocol stack.

The following are some protocols for VOIP

Megaco H.248 :

Gateway Control Protocol

MGCP:

Media Gateway Control Protocol

MIME :

RVP over IP:

Remote Voice Protocol Over IP Specification

SAPv2 :

Session Announcement Protocol

SDP :

Session Description Protocol

SGCP :

Simple Gateway Control Protocol

SIP :

Session Initiation Protocol

Skinny :

Skinny Client Control Protocol (SCCP)

Session Initiation Protocol

Session initiation protocol is a widely used application layer based protocol with simple signaling. SIP gateways are essential in VoIP networks as they allow for multiple audio and video connections to take place at one time over the Internet. Moreover, using a SIP gateway is an affordable and efficient method of networking in and outside office. SIP provides many facilities like call forwarding, Caller number identification, caller & callee identification, Conference calling and instant messaging.

Testing

The SIP (session initiation protocol) signaling protocol is emerging as the industry's preferred Voice over IP (VoIP) technology. As such, SIP's scalability is key to VoIP technology's continued adoption, and the need to test SIP user agents and servers under heavily saturated scenarios is becoming more acute. SIP servers must be able to bear heavy traffic loads for user registration and call setup functions that require reading from and writing to one or

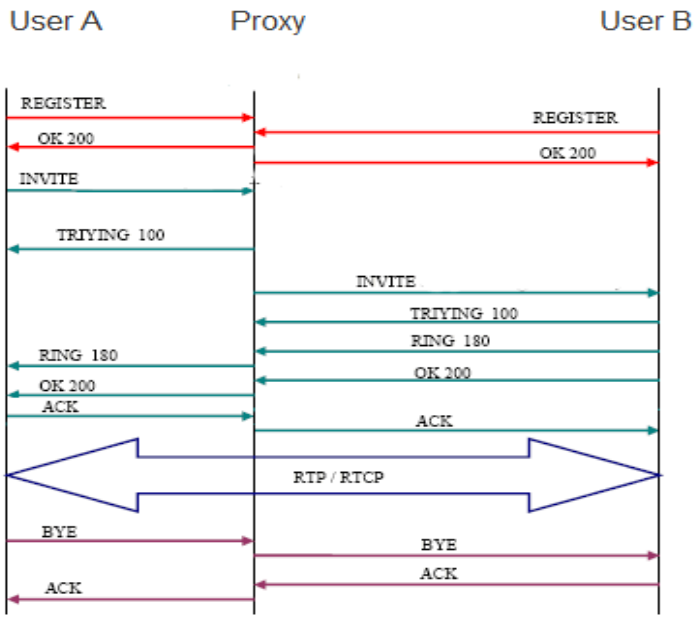
more large dynamic database(s). The corresponding test tools, call generators, terminators and analyzers must all be capable of processing the same significant loads. Data packet analyzers are used to see and analyze the sip related packets flowing from the application which help to troubleshoot the issue raised. The issues based on packets can be easily understood by seeing them in a packet analyzer tool.

Let us take some simple examples

Scenario1: -Suppose that there are two ends, end “A” and end “B” and an SIP server. Both end “A” and end “B” have user agents (Software applications) at their ends and registered to SIP server. End “A” wants to communicate with end “B” and initiated a call to end “B” but not able to connect with end “B” even after dialing correct number .Here for analyzing the problem we will open data packet analyzer tool at end “A” and will look where the problem is? Now we again try to connect end “A” by initiating a call from end “A” to end “B”. In data packet analyzer SIP signals showed which are initiated from user agent of end “A”. We can see that with the number to which end “A” initiated call, tool showing some special characters. It means may be End “A” inserting space in after number and trying to connect. User agent should ignore the space before and after the number dialed but it is not doing so, which was the actual problem.

We see how a tester can test a application based on VOIP. Similarly other possible scenarios are as follows. Assuming the same setup we have in the above example.

Scenario 2: While testing we need to ensure that the basic functionalities of application are working properly and will also check the packet flow. Let us assume end “A” and “B” are registered in SIP server. When end “A” initiates a call to end “B” data packets travel between both ends, proxy server plays the role of intermediary between both ends. The basic call flow is showed in following diagram. QA engineer compares the expected packet flow with actual packet flow. If there is any deviation and its side effect reflects on application then you have found a bug.



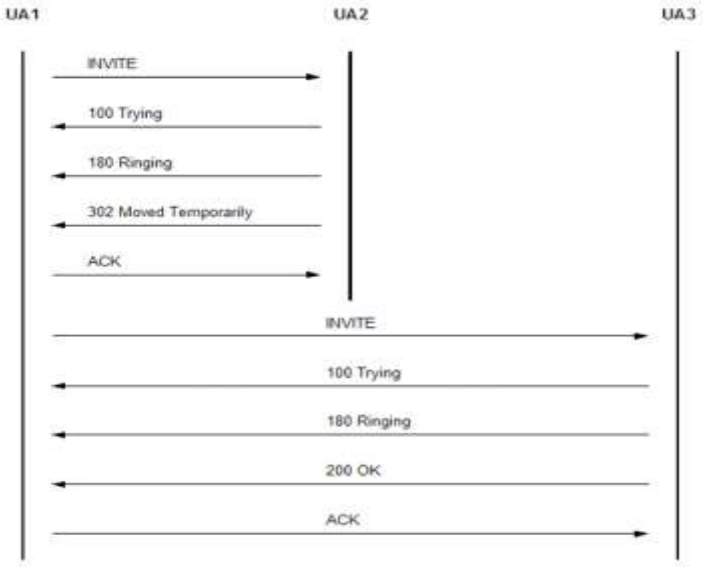
Call flow of Session initiation protocol

Call initiation: When “A” initiates call to “B” an invite packet is sent from “A” to server, server will reply by sending “100 trying” packet and forward invite packet to “B”. End “B” also reply to sever by “100 trying” packet.

Call Ringing state: when call is in ringing state, end “B” will send “100 ringing” packet to server, and server forward it to “A”. Now when “B” answers the call and it will send “200 ok” to server and server will forward it to “A”. “A” will send ACK packet to server and “B”.

Call session and termination: After answering the call. A data stream will then established between “A” and “B”. Packets are bypassed by server and can sent and received directly by both ends. After ending of session a “bye” packet will flow and remote end will acknowledge it by sending ACK packet.

Scenario 3: We saw two scenarios related to basic call functionality and their packet flow. In this example I would like to tell about call forwarding functionalities of the application. End “A” initiates call to “B”. End “B” can forward the incoming call to another end C. Following diagram is showing the packet flow for forwarding a call.



After hitting forward button by end B, a “302 moved temporarily” packet will flow from “B” to “A“ to say end A to forward call at end C. After receiving “302”packet from “B”, “A” will send acknowledge to B and send an invite packet to end C. After this it will follow the basic flow of call.

In this way we do VOIP protocol testing. Seems interesting isn’t it?

Beyond TESTING

find us on facebook & twitter

http://facebook.com/beyondtestingmag

http://twitter.com/beyondtesting