About "CHETANA-JOBS" & "ChetanaS" (www.ChetanaS.com)

"**CHETANA-JOBS**" is **World's Biggest Job Group** with more than **2 LAKH** members and through which more than **40,000** job seekers got jobs till now. In fact, Chetana created a history by becoming World's Biggest Job Group just within 2 years of establishment. Since the beginning, Chetana is committed to provide the Job Seekers with the latest job information and the proper career guidance they need for their job search. The journey of Chetana to help others is still continuing...and will continue...

URL of "CHETANA-JOBS": **http://groups.yahoo.com/group/CHETANA-JOBS**

Official Job WebSite: **www.ChetanaS.com**

(Important Note: Please note that **www.ChetanaS.com** is the only official job website of "CHETANA-JOBS" Yahoo Group. There are many websites trying to use the name of 'Chetana' to get the brand value to their Job groups or WebSites and to mislead people. And there are many websites that copy the job information from *www.ChetanaS.com* and display in their websites/groups. Please be aware of that. If you get such doubt on any site, then please avoid visiting them and save your precious time and money.)

Please visit **www.ChetanaS.com** for latest WALK-IN / OFF-CAMPUS job information, Fresher Jobs, Experienced Jobs, Job Search Tips, Placement Papers, Resume Writing Tips, Test/Interview/GD Guidelines, Motivational Thoughts, Success Stories and much more...

TESTING - FAQ

What makes a good test engineer?

A good test engineer has a 'test to break' attitude, an ability to take the point of view of the customer, a strong desire for quality, and an attention to detail.
Tact and diplomacy are useful in maintaining a cooperative relationship with developers, and an ability to communicate with both technical (developers) and non-technical (customers, management) people is useful.
Previous software development experience can be helpful as it provides a deeper understanding of the software development process, gives the tester an appreciation for the developers' point of view, and reduce the learning curve in automated test tool programming.
Judgement skills are needed to assess high-risk areas of an application on which to focus testing efforts when time is limited.

What makes a good software QA engineer?

The same qualities a good tester has are useful for a QA engineer.
Additionally, they must be able to understand the entire software development process and how it can fit into the business approach and goals of the organization.
Communication skills and the ability to understand various sides of issues are important.
In organizations in the early stages of implementing QA processes, patience and diplomacy are especially needed.
An ability to find problems as well as to see 'what's missing' is important for inspections and reviews.

What makes a good QA or Test manager?

A good QA, test, or QA/Test (combined) manager should:
be familiar with the software development process
be able to maintain enthusiasm of their team and promote a positive atmosphere, despite what is a somewhat 'negative' process (e.g., looking for or preventing problems)
be able to promote teamwork to increase productivity
be able to promote cooperation between software, test, and QA engineers
have the diplomatic skills needed to promote improvements in QA processes
have the ability to withstand pressures and say 'no' to other managers when quality is insufficient or QA processes are not being adhered to
have people judgement skills for hiring and keeping skilled personnel
be able to communicate with technical and non-technical people, engineers, managers, and customers.
be able to run meetings and keep them focused

What's the role of documentation in QA?

Critical. (Note that documentation can be electronic, not necessarily paper.)
QA practices should be documented such that they are repeatable.
Specifications, designs, business rules, inspection reports, configurations, code changes, test plans, test cases, bug reports, user manuals, etc. should all be documented.
There should ideally be a system for easily finding and obtaining documents and determining what documentation will have a particular piece of information.
Change management for documentation should be used if possible.

What's the big deal about 'requirements'?

One of the most reliable methods of insuring problems, or failure, in a complex software project is to have poorly documented requirements specifications.
Requirements are the details describing an application's externally-perceived functionality and properties.
Requirements should be clear, complete, reasonably detailed, cohesive, attainable, and testable. A non-testable requirement would be, for example, 'user-friendly' (too subjective).
A testable requirement would be something like 'the user must enter their previously-assigned password to access the application'.
Determining and organizing requirements details in a useful and efficient way can be a difficult effort; different methods are available depending on the particular project.
Many books are available that describe various approaches to this task.
Care should be taken to involve ALL of a project's significant 'customers' in the requirements process.
'Customers' could be in-house personnel or out, and could include end-users, customer acceptance testers, customer contract officers, customer management, future software maintenance engineers, salespeople, etc.
Anyone who could later derail the project if their expectations aren't met should be included if possible.
Organizations vary considerably in their handling of requirements specifications.
Ideally, the requirements are spelled out in a document with statements such as 'The product shall.'.
'Design' specifications should not be confused with 'requirements'; design specifications should be traceable back to the requirements.
In some organizations requirements may end up in high level project plans, functional specification documents, in design documents, or in other documents at various levels of detail.
No matter what they are called, some type of documentation with detailed requirements will be needed by testers in order to properly plan and execute tests.
Without such documentation, there will be no clear-cut way to determine if a software application is performing correctly.

What steps are needed to develop and run software tests?

The following are some of the steps to consider:
Obtain requirements, functional design, and internal design specifications and other necessary documents
Obtain budget and schedule requirements
Determine project-related personnel and their responsibilities, reporting requirements, required standards and processes (such as release processes, change processes, etc.)
Identify application's higher-risk aspects, set priorities, and determine scope and limitations of tests
Determine test approaches and methods - unit, integration, functional, system, load, usability tests, etc.
Determine test environment requirements (hardware, software, communications, etc.)
Determine testware requirements (record/playback tools, coverage analyzers, test tracking, problem/bug tracking, etc.)
Determine test input data requirements
Identify tasks, those responsible for tasks, and labor requirements
Set schedule estimates, timelines, milestones
Determine input equivalence classes, boundary value analyses, error classes
Prepare test plan document and have needed reviews/approvals
Write test cases
Have needed reviews/inspections/approvals of test cases
Prepare test environment and testware, obtain needed user manuals/reference documents/configuration guides/installation guides, set up test tracking processes, set up logging and archiving processes, set up or obtain test input data
Obtain and install software releases
Perform tests

Evaluate and report results
Track problems/bugs and fixes
Retest as needed
Maintain and update test plans, test cases, test environment, and testware through life cycle

What's a 'test plan'?

A software project test plan is a document that describes the objectives, scope, approach, and focus of a software testing effort. The process of preparing a test plan is a useful way to think through the efforts needed to validate the acceptability of a software product. The completed document will help people outside the test group understand the 'why' and 'how' of product validation. It should be thorough enough to be useful but not so thorough that no one outside the test group will read it. The following are some of the items that might be included in a test plan, depending on the particular project:
Title
Identification of software including version/release numbers
Revision history of document including authors, dates, approvals
Table of Contents
Purpose of document, intended audience
Objective of testing effort
Software product overview
Relevant related document list, such as requirements, design documents, other test plans, etc.
Relevant standards or legal requirements
Traceability requirements
Relevant naming conventions and identifier conventions
Overall software project organization and personnel/contact-info/responsibilties
Test organization and personnel/contact-info/responsibilities
Assumptions and dependencies
Project risk analysis
Testing priorities and focus
Scope and limitations of testing
Test outline - a decomposition of the test approach by test type, feature, functionality, process, system, module, etc. as applicable
Outline of data input equivalence classes, boundary value analysis, error classes
Test environment - hardware, operating systems, other required software, data configurations, interfaces to other systems
Test environment setup and configuration issues
Test data setup requirements
Database setup requirements
Outline of system-logging/error-logging/other capabilities, and tools such as screen capture software, that will be used to help describe and report bugs
Discussion of any specialized software or hardware tools that will be used by testers to help track the cause or source of bugs
Test automation - justification and overview
Test tools to be used, including versions, patches, etc.
Test script/test code maintenance processes and version control
Problem tracking and resolution - tools and processes
Project test metrics to be used
Reporting requirements and testing deliverables

Software entrance and exit criteria
Initial sanity testing period and criteria
Test suspension and restart criteria
Personnel allocation
Personnel pre-training needs
Test site/location
Outside test organizations to be utilized and their purpose, responsibilties, deliverables, contact persons, and coordination issues
Relevant proprietary, classified, security, and licensing issues.
Open issues
Appendix - glossary, acronyms, etc.

What's a 'test case'?
A test case is a document that describes an input, action, or event and an expected response, to determine if a feature of an application is working correctly.
A test case should contain particulars such as test case identifier, test case name, objective, test conditions/setup, input data requirements, steps, and expected results.
Note that the process of developing test cases can help find problems in the requirements or design of an application, since it requires completely thinking through the operation of the application. For this reason, it's useful to prepare test cases early in the development cycle if possible.

What should be done after a bug is found?
The bug needs to be communicated and assigned to developers who can fix it.
After the problem is resolved, fixes should be re-tested, and determinations made regarding requirements for regression testing to check that fixes didn't create problems elsewhere. If a problem-tracking system is in place, it should encapsulate these processes. A variety of commercial problem-tracking/management software tools are available . The following are items to be considered in the tracking process:
Complete information such that developers can understand the bug, get an idea of it's severity, and reproduce it if necessary.
Bug identifier (number, ID, etc.)
Current bug status (e.g., 'Released for Retest', 'New', etc.)
The application name or identifier and version
The function, module, feature, object, screen, etc. where the bug occurred
Environment specifics, system, platform, relevant hardware specifics
Test case name/number/identifier
One-line bug description
Full bug description
Description of steps needed to reproduce the bug if not covered by a test case or if the developer doesn't have easy access to the test case/test script/test tool
Names and/or descriptions of file/data/messages/etc. used in test
File excerpts/error messages/log file excerpts/screen shots/test tool logs that would be helpful in finding the cause of the problem
Severity estimate (a 5-level range such as 1-5 or 'critical'-to-'low' is common)
Was the bug reproducible?
Tester name
Test date
Bug reporting date
Name of developer/group/organization the problem is assigned to

Description of problem cause
Description of fix
Code section/file/module/class/method that was fixed
Date of fix
Application version that contains the fix
Tester responsible for retest
Retest date
Retest results
Regression testing requirements
Tester responsible for regression tests
Regression testing results
A reporting or tracking process should enable notification of appropriate personnel at various stages.
For instance, testers need to know when retesting is needed, developers need to know when bugs are found and how to get the needed information, and reporting/summary capabilities are needed for managers.

What is 'configuration management'?

Configuration management covers the processes used to control, coordinate, and track: code, requirements, documentation, problems, change requests, designs, tools/compilers/libraries/patches, changes made to them, and who makes the changes.

What if the software is so buggy it can't really be tested at all?

The best bet in this situation is for the testers to go through the process of reporting whatever bugs or blocking-type problems initially show up, with the focus being on critical bugs. Since this type of problem can severely affect schedules, and indicates deeper problems in the software development process (such as insufficient unit testing or insufficient integration testing, poor design, improper build or release procedures, etc.) managers should be notified, and provided with some documentation as evidence of the problem.

How can it be known when to stop testing?

This can be difficult to determine. Many modern software applications are so complex, and run in such an interdependent environment, that complete testing can never be done. Common factors in deciding when to stop are:
Deadlines (release deadlines, testing deadlines, etc.)
Test cases completed with certain percentage passed
Test budget depleted
Coverage of code/functionality/requirements reaches a specified point
Bug rate falls below a certain level
Beta or alpha testing period ends

What if there isn't enough time for thorough testing?
Use risk analysis to determine where testing should be focused.

Since it's rarely possible to test every possible aspect of an application, every possible combination of events, every dependency, or everything that could go wrong, risk analysis is appropriate to most software development projects. This requires judgement skills, common sense, and experience. (If warranted, formal methods are also available.)
Considerations can include:
Which functionality is most important to the project's intended purpose?

Which functionality is most visible to the user?
Which functionality has the largest safety impact?
Which functionality has the largest financial impact on users?
Which aspects of the application are most important to the customer?
Which aspects of the application can be tested early in the development cycle?
Which parts of the code are most complex, and thus most subject to errors?
Which parts of the application were developed in rush or panic mode?
Which aspects of similar/related previous projects caused problems?
Which aspects of similar/related previous projects had large maintenance expenses?
Which parts of the requirements and design are unclear or poorly thought out?
What do the developers think are the highest-risk aspects of the application?
What kinds of problems would cause the worst publicity?
What kinds of problems would cause the most customer service complaints?
What kinds of tests could easily cover multiple functionalities?
Which tests will have the best high-risk-coverage to time-required ratio?

What if the project isn't big enough to justify extensive testing?

Consider the impact of project errors, not the size of the project.
However, if extensive testing is still not justified, risk analysis is again needed and the same considerations as described previously in.
The tester might then do ad hoc testing, or write up a limited test plan based on the risk analysis.

What can be done if requirements are changing continuously?

A common problem and a major headache.
Work with the project's stakeholders early on to understand how requirements might change so that alternate test plans and strategies can be worked out in advance, if possible.
It's helpful if the application's initial design allows for some adaptability so that later changes do not require redoing the application from scratch.
If the code is well-commented and well-documented this makes changes easier for the developers.
Use rapid prototyping whenever possible to help customers feel sure of their requirements and minimize changes.
The project's initial schedule should allow for some extra time commensurate with the possibility of changes.
Try to move new requirements to a 'Phase 2' version of an application, while using the original requirements for the 'Phase 1' version.
Negotiate to allow only easily-implemented new requirements into the project, while moving more difficult new requirements into future versions of the application.
Be sure that customers and management understand the scheduling impacts, inherent risks, and costs of significant requirements changes. Then let management or the customers (not the developers or testers) decide if the changes are warranted - after all, that's their job.
Balance the effort put into setting up automated testing with the expected effort required to re-do them to deal with changes.
Try to design some flexibility into automated test scripts.
Focus initial automated testing on application aspects that are most likely to remain unchanged.
Devote appropriate effort to risk analysis of changes to minimize regression testing needs.
Design some flexibility into test cases (this is not easily done; the best bet might be to minimize the detail in the test cases, or set up only higher-level generic-type test plans)
Focus less on detailed test plans and test cases and more on ad hoc testing (with an understanding of the added risk that

this entails).

What if the application has functionality that wasn't in the requirements?

It may take serious effort to determine if an application has significant unexpected or hidden functionality, and it would indicate deeper problems in the software development process.

If the functionality isn't necessary to the purpose of the application, it should be removed, as it may have unknown impacts or dependencies that were not taken into account by the designer or the customer.
If not removed, design information will be needed to determine added testing needs or regression testing needs.
Management should be made aware of any significant added risks as a result of the unexpected functionality.
If the functionality only effects areas such as minor improvements in the user interface, for example, it may not be a significant risk.

How can Software QA processes be implemented without stifling productivity?

By implementing QA processes slowly over time, using consensus to reach agreement on processes, and adjusting and experimenting as an organization grows and matures, productivity will be improved instead of stifled. Problem prevention will lessen the need for problem detection, panics and burn-out will decrease, and there will be improved focus and less wasted effort. At the same time, attempts should be made to keep processes simple and efficient, minimize paperwork, promote computer-based processes and automated tracking and reporting, minimize time required in meetings, and promote training as part of the QA process. However, no one - especially talented technical types - likes rules or bureacracy, and in the short run things may slow down a bit. A typical scenario would be that more days of planning and development will be needed, but less time will be required for late-night bug-fixing and calming of irate customers.

What if an organization is growing so fast that fixed QA processes are impossible?

This is a common problem in the software industry, especially in new technology areas. There is no easy solution in this situation, other than:
Hire good people
Management should 'ruthlessly prioritize' quality issues and maintain focus on the customer
Everyone in the organization should be clear on what 'quality' means to the customer

How does a client-server environment affect testing?

Client/server applications can be quite complex due to the multiple dependencies among clients, data communications, hardware, and servers.
Thus testing requirements can be extensive. When time is limited (as it usually is) the focus should be on integration and system testing.
Additionally, load/stress/performance testing may be useful in determining client/server application limitations and capabilities.
There are commercial tools to assist with such testing.

How can World Wide Web sites be tested?

Web sites are essentially client/server applications - with web servers and 'browser' clients. Consideration should be given to the interactions between html pages, TCP/IP communications, Internet connections, firewalls, applications that

run in web pages (such as applets, javascript, plug-in applications), and applications that run on the server side (such as cgi scripts, database interfaces, logging applications, dynamic page generators, asp, etc.).
Additionally, there are a wide variety of servers and browsers, various versions of each, small but sometimes significant differences between them, variations in connection speeds, rapidly changing technologies, and multiple standards and protocols.
The end result is that testing for web sites can become a major ongoing effort.
Other considerations might include:
What are the expected loads on the server (e.g., number of hits per unit time?), and what kind of performance is required under such loads (such as web server response time, database query response times).
What kinds of tools will be needed for performance testing (such as web load testing tools, other tools already in house that can be adapted, web robot downloading tools, etc.)?
Who is the target audience? What kind of browsers will they be using? What kind of connection speeds will they by using? Are they intra- organization (thus with likely high connection speeds and similar browsers) or Internet-wide (thus with a wide variety of connection speeds and browser types)?
What kind of performance is expected on the client side (e.g., how fast should pages appear, how fast should animations, applets, etc. load and run)?
Will down time for server and content maintenance/upgrades be allowed? how much?
What kinds of security (firewalls, encryptions, passwords, etc.) will be required and what is it expected to do? How can it be tested?
How reliable are the site's Internet connections required to be? And how does that affect backup system or redundant connection requirements and testing?
What processes will be required to manage updates to the web site's content, and what are the requirements for maintaining, tracking, and controlling page content, graphics, links, etc.?
Which HTML specification will be adhered to? How strictly? What variations will be allowed for targeted browsers?
Will there be any standards or requirements for page appearance and/or graphics throughout a site or parts of a site??
How will internal and external links be validated and updated? how often?
Can testing be done on the production system, or will a separate test system be required? How are browser caching, variations in browser option settings, dial-up connection variabilities, and real-world internet 'traffic congestion' problems to be accounted for in testing?
How extensive or customized are the server logging and reporting requirements; are they considered an integral part of the system and do they require testing?
How are cgi programs, applets, javascripts, ActiveX components, etc. to be maintained, tracked, controlled, and tested?

Pages should be 3-5 screens max unless content is tightly focused on a single topic. If larger, provide internal links within the page.
The page layouts and design elements should be consistent throughout a site, so that it's clear to the user that they're still within a site.
Pages should be as browser-independent as possible, or pages should be provided or generated based on the browser-type.
All pages should have links external to the page; there should be no dead-end pages.
The page owner, revision date, and a link to a contact person or organization should be included on each page.

How is testing affected by object-oriented designs?

Well-engineered object-oriented design can make it easier to trace from code to internal design to functional design to requirements.
While there will be little affect on black box testing (where an understanding of the internal design of the application is unnecessary), white-box testing can be oriented to the application's objects. If the application was well-designed this can

simplify test design.

What is Extreme Programming and what's it got to do with testing ?

Extreme Programming (XP) is a software development approach for small teams on risk-prone projects with unstable requirements.
It was created by Kent Beck who described the approach in his book 'Extreme Programming Explained' .
Testing ('extreme testing') is a core aspect of Extreme Programming.
Programmers are expected to write unit and functional test code first - before the application is developed.
Test code is under source control along with the rest of the code.
Customers are expected to be an integral part of the project team and to help develop scenarios for acceptance/black box testing.
Acceptance tests are preferably automated, and are modified and rerun for each of the frequent development iterations.

QA and test personnel are also required to be an integral part of the project team.

Detailed requirements documentation is not used, and frequent re-scheduling, re-estimating, and re-prioritizing is expected

Contributed By: Priya Nair (CHETANAS Member)

Special Notes

The information in this document will be updated time to time based on the feedback we get from various sources. At any time, you can visit our website (www.ChetanaS.com) and get the latest version of this document/other documents by observing the Version Number.

Please visit www.ChetanaS.com for latest WALK-IN / OFF-CAMPUS job information, Fresher Jobs, Experienced Jobs, Job Search Tips, Placement Papers, Resume Writing Tips, Test/Interview/GD Guidelines, Motivational Thoughts, Success Stories and much more...

If you have any copyright issues regarding the information displayed in this document, please mail me complete details at: Chetana@ChetanaS.com