**What makes a good Software 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, product owners) 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 enhances automated test programming skills. Judgement skills are needed to assess high-risk or critical areas of an application on which to focus testing efforts when time is limited. In recent years the role of the software test engineer has been in flux, and in some organizations test engineers are more technical, being also involved in developing or maintaining continuous integration and delivery processes, and/or developing test automation capabilities and integrating them into these processes.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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 provide appropriate feedback to other managers when there are issues with quality/processes/schedules/risk
* 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

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What's the role of documentation in QA?**  
Generally, the larger the team/organization, the more useful it will be to stress documentation, in order to manage and communicate more efficiently. (Note that documentation may be electronic, not necessarily in printable form, and may be embedded in code comments, may be embodied in well-written test cases, user stories, etc.) QA practices may be documented to enhance their repeatability. Specifications, designs, business rules, configurations, code changes, test plans, test cases, bug reports, user manuals, etc. may be documented in some form. There would ideally be a system for easily finding and obtaining information and determining what documentation will have a particular piece of information. Change management for documentation can be used where appropriate. For agile software projects, it should be kept in mind that one of the agile values is "Working software over comprehensive documentation", which does not mean 'no' documentation. Agile projects tend to stress the short term view of project needs; documentation often becomes more important in a project's long-term context.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What's the big deal about 'requirements'?**  
Depending on the project, it may or may not be a 'big deal'. For agile projects, which may be more amenable to changing requirements, detailed documented requirements may not be needed. However some type of documented specfications are still important, in the form of user stories or something similar. For non-agile types of projects detailed documented requirements are usually needed. (Note that requirements documentation can be electronic, not necessarily in the form of printable documents, and may be embedded in code comments, or may be embodied in well-written test cases, wiki's, user stories, etc.) Requirements are the details describing an application's externally-perceived functionality and properties. Requirements are ideally clear, complete, reasonably detailed, cohesive, attainable, and testable. A non-testable requirement would be, for example, 'user-friendly' (too subjective). A more 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 and software tools are available depending on the particular project. Many books are available that describe various approaches to this task, for either agile or non-agile contexts. (See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Requirements and User Stories' category.)

Care should be taken to involve ALL of a project's relevant 'customers' in the requirements/user story process. 'Customers' could be in-house personnel or outside personnel, 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 success of the project if their expectations aren't met should be included if possible. In agile projects, a product owner is often considered the representative of all 'customers', but in some cases a single product owner may not be the best approach and it may be more appropriate to involve other stakeholders more directly.

Organizations vary considerably in their handling of requirements specifications. In agile projects, some or all requirements may be embodied in user stories. In other projects the requirements may be spelled out in a document with statements such as 'The product shall.....'. 'Design' specifications should not be confused with 'requirements'. In some contexts it can be helpful to have design specifications (if any) traceable back to the requirements.

In some organizations requirements may end up in high level project plans, functional specification documents, in design documents, user stories, or in other documents at various levels of detail. No matter what they are called, some type of documentation with specifications and related information will be useful to testers in order to properly plan and execute tests (manual or automated). Without such documentation, there will be no clear-cut way to determine if software is working as expected.

If testable requirements are not available or are only partially available, useful testing can still be performed. In this situation test results may be more oriented to providing information about the state of the software and risk levels, rather than providing pass/fail results. A relevant testing approach in this situation may include approaches such as 'exploratory testing'. Many software projects have a mix of user stories, documented testable requirements, poorly documented requirements, undocumented requirements, and changing requirements. In such projects a mix of automated,scripted, and exploratory testing approaches may be useful. See the Softwareqatest.com 'Other Resources' page in the ['General Software QA and Testing Resources' section](http://www.softwareqatest.com/qatlnks1.html#GENERAL)for articles on exploratory testing, and in the ['Agile and XP Testing Resources' section](http://www.softwareqatest.com/qatlnks1.html#AGILE)for articles on agile software development and testing.)

'Agile' approaches require close interaction and cooperation between development teams and stakeholders/customers/end-users to iteratively develop requirements, user stories, etc. In the XP 'test first' approach developers create automated unit testing code before the application code, and these automated unit tests could essentially embody requirements.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What steps are needed to develop and run software tests?**  
The following are some of the steps to consider, depending on the project context (large, small, agile, non-agile, etc): (Note: these apply to an overall testing approach or manual testing approach; for more information on automated testing see the [SoftwareQATest.com LFAQ page.)](http://www.softwareqatest.com/qat_lfaq1.html#LFAQ1_8)

* Obtain user stories, requirements, functional design, internal design specifications, or other available/necessary information
* 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.)
* Determine project context, relative to the existing quality culture of the product/organization/business, and how it might impact testing scope, approaches, and methods.
* Identify the application's higher-risk and more important aspects, set priorities, and determine scope and limitations of tests.
* Determine test approaches and methods - unit, integration, functional, system, security, load, usability tests, whichever are in scope.
* Determine test environment requirements (hardware, software, configuration, versions, communications, etc.)
* Determine testware requirements (automation tools, coverage analyzers, test tracking, problem/bug tracking, etc.)
* Determine test input data requirements
* Identify tasks, those responsible for tasks, and labor requirements
* Set initial schedule estimates, timelines, milestones where feasible.
* Determine, where appropriate, input equivalence classes, boundary value analyses, error classes
* Prepare test plan document(s) and have needed reviews/approvals
* Write test cases or test scenarios as needed.
* Have needed reviews/inspections/approvals of test cases/scenarios/approaches.
* 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/install/configure 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

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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 overly detailed 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/responsibilities
* 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 validity analysis - differences between the test and production systems and their impact on test validity.
* Test environment setup and configuration issues
* Software migration processes
* Software CM processes
* 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.

(See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software Testing' and 'Software QA' categories for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What's a 'test case'?**  
A test case describes an input, action, or event and an expected response, to determine if a feature of a software application is working correctly. A test case may contain particulars such as test case identifier, test case name, objective, test conditions/setup, input data requirements, steps, and expected results. The level of detail may vary significantly depending on the organization and project context. Note that organizations vary considerably in their handling of test cases; many utilize less-detailed 'test scenarios' that allow for simpler and more adaptable/maintainable test documentation, many also use BDD-style test scenarios using the Gherkin syntax.

Note that the process of developing test cases can help find problems in the requirements/user stories/design of an application, since it requires thinking through the operation of the application. For this reason, it's useful to prepare test cases early in the development cycle if possible.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What should be done after a bug is found?**  
The bug needs to be communicated and assigned to developers that 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 used, it should encapsulate these processes. The following are items to consider 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 or scenario information/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 automated test 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.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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. (See the ['Tools'](http://www.softwareqatest.com/qattls1.html) section for web resources with listings of configuration management tools. Also see the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Configuration Management' category for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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 significantly 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.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**How can it be known when to stop testing?**  
This can be difficult to determine. Most 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/user stories/criteria reaches a specified point
* Bug rate falls below a certain level
* Beta or alpha testing period ends

Also see ['Who should decide when software is ready to be released?'](http://www.softwareqatest.com/qat_lfaq1.html#LFAQ1_3) in the LFAQ section.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What if there isn't enough time for thorough testing?**  
Use risk analysis, along with discussion with project stakeholders, 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?

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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 ['What if there isn't enough time for thorough testing?'](http://www.softwareqatest.com/qatfaq2.html#FAQ2_12) apply. The tester might then do ad hoc or exploratory testing, or write up a limited test plan based on the risk analysis.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**How do distributed multi-tier environments affect testing?**  
Most current software being tested involves multi-tier and distributed applications which can be highly complex due to the multiple dependencies among systems, services, data communications, hardware, and servers. Thus testing requirements can be extensive. When time is limited (as it usually is) a focus on integration and system testing can be considered. Additionally, load/stress/performance testing may be useful in determining distributed application limitations and capabilities and where the limitations are. There are commercial and open source tools to assist with such testing. (See the ['Tools'](http://www.softwareqatest.com/qattls1.html) section for web resources with listings that include these kinds of test tools.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**How should Web sites be tested?**  
Many modern web sites are essentially complex distributed systems with html, css, web services, microservices, encrypted communications, browser-side scripts/apps/libraries (such as javascript, flash, etc), the wide variety of applications/libraries/datastores that could run on the server side, load balancers, content delivery networks, etc. Additionally, there are a wide variety of servers and browsers, mobile and other platforms, various versions of each, small but sometimes significant differences between them, variations in connection speeds, rapidly changing technologies, and multiple standards and protocols. Although web site testing was initially relatively simple years ago, testing of modern web site front ends, back end systems, mid-level tiers, web services, databases, security, performance, etc, can be as complex as or more complex than any other type of application.

To assist in testing of web sites via their GUI's, many popular web browsers normally include a set of 'Developer Tools' that are helpful in testing and debugging, and in developing test automation scripts:

* [Chrome DevTools](https://developer.chrome.com/devtools)
* [Firefox Developer Tools](https://developer.mozilla.org/en-US/docs/Tools)
* [IE - Using the F12 Developer Tools](http://msdn.microsoft.com/en-us/library/ie/bg182326(v=vs.85).aspx)
* [Safari for developers - Web Inspector](https://developer.apple.com/safari/tools/)

For more information see the various web-related testing resource sections (Mobile Testing Resources, Web QA and Testing Resources, Web Security Testing Resources, Web Usability Resources, etc.) in the SoftwareQATest.com ['Other Resources' section.](http://www.softwareqatest.com/qatlnks1.html)

Hundreds of web site test tools are available and more than 560 of them, in 14 categories, are listed in the ['Web Test Tools' section](http://www.softwareqatest.com/qatweb1.html).

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**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 effect 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, methods, etc. If the application was well-designed this can simplify test design and test automation design.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq2.html#TOP)

**What is Agile Software Development and how does it impact testing?**  
Agile Software Development generally refers to incremental, collaborative software development approaches that provide alternatives to 'heavyweight', documentation-driven, waterfall-type development practices. It grew out of such approaches as Extreme Programming, SCRUM, DSDM, Crystal, and other 'lightweight' methodologies. In 2001 a group of software development and test practitioners gathered to discuss lightweight methods and created the ['Agile Manifesto'](http://agilemanifesto.org/)which describes the Agile approach values and lists 12 principles that describe Agile software development. In reality many organizations implement these principles to widely varying degrees (and with widely varying degrees of success) and still call their approach 'Agile'.   
The impact of Agile approaches on software testing can also vary widely but often includes the following:

* Requirements and documentation are often minimal and when present are often in the form of high-level 'user stories' and 'acceptance criteria/tests'.
* Requirements can be added or changed often
* Iterative development/test cycles ('sprints') are often in the range of 1-3 weeks. Both new functionality testing and regression testing (preferably automated) may occur within each iterative cycle.
* Close collaboration between testers and developers, product owners and other team members
* Short daily project status 'standup' meetings that include testers.
* Common testing-related practices in agile projects may include test-driven development, extensive unit testing and unit test automation, API-level test automation, exploratory and session-based testing, continuous integration, UI test automation.
* Testers may be heavily involved in fleshing out requirements/criteria details, including both functional and non-functional requirements.