Quality Assurance Engineering

Angelique Mitchel

First Edition, 2012

ISBN 978-81-323-4013-3

Published by:
White Word Publications
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter-1

Quality Assurance

Quality assurance, or **QA** for short, is the systematic monitoring and evaluation of the various aspects of a project, service or facility to maximize the probability that minimum standards of quality are being attained by the production process. QA cannot absolutely guarantee the production of *quality* products.

Two principles included in QA are: "Fit for purpose" - the product should be suitable for the intended purpose; and "Right first time" - mistakes should be eliminated. QA includes regulation of the quality of raw materials, assemblies, products and components, services related to production, and management, production and inspection processes.

Quality is determined by the product users, clients or customers, not by society in general. It is not the same as 'expensive' or 'high quality'. Low priced products can be considered as having high quality if the product users determine them as such.

Early efforts to control the quality of production

During the Middle Ages, guilds adopted responsibility for quality control of their members, setting and maintaining certain standards for guild membership.

Royal governments purchasing material were interested in quality control as customers. For this reason, King John of England appointed William Wrotham to report about the construction and repair of ships. Centuries later, Samuel Pepys, Secretary to the British Admiralty, appointed multiple such overseers.

Prior to the extensive division of labor and mechanization resulting from the Industrial Revolution, it was possible for workers to control the quality of their own products. The Industrial Revolution led to a system in which large groups of people performing a similar type of work were grouped together under the supervision of a foreman who was appointed to control the quality of work manufactured.

Wartime production

At the time of the First World War, manufacturing processes typically became more complex with larger numbers of workers being supervised. This period saw the widespread introduction of mass production and piecework, which created problems as workmen could now earn more money by the production of extra products, which in turn occasionally led to poor quality workmanship being passed on to the assembly lines. To counter bad workmanship, full time inspectors were introduced into the to identify, quarantine and ideally correct product quality failures. Quality control by inspection in the 1920s and 1930s led to the growth of quality inspection functions, separately organised from production and large enough to be headed by superintendents.

The systematic approach to quality started in industrial manufacture during the 1930s, mostly in the USA, when some attention was given to the cost of scrap and rework. With the impact of mass production required during the Second World War made it necessary to introduce an improved form of quality control known as Statistical Quality Control, or SQC. Some of the initial work for SQC is credited to Walter A. Shewhart of Bell Labs, starting with his famous one-page memorandum of 1924.

SQC includes the concept that every production piece cannot be fully inspected into acceptable and nonacceptable batches. By extending the inspection phase and making inspection organizations more efficient, it provides inspectors with control tools such as sampling and control charts, even where 100 per cent inspection is not practicable. Standard statistical techniques allow the producer to sample and test a certain proportion of the products for quality to achieve the desired level of confidence in the quality of the entire batch or production run.

Postwar

In the period following World War II, many countries' manufacturing capabilities that had been destroyed during the war were rebuilt. General Douglas MacArthur oversaw the re-building of Japan. During this time, General MacArthur involved two key individuals in the development of modern quality concepts: W. Edwards Deming and Joseph Juran. Both individuals promoted the collaborative concepts of quality to Japanese business and technical groups, and these groups utilized these concepts in the redevelopment of the Japanese economy.

Although there were many individuals trying to lead United States industries towards a more comprehensive approach to quality, the U.S. continued to apply the Quality Control (QC) concepts of inspection and sampling to remove defective product from production lines, essentially ignoring advances in QA for decades.

Steps for a typical quality assurance process

There are many forms of QA processes, of varying scope and depth. The application of a particular process is often customized to the production process.

A typical process may include:

- test of previous articles
- plan to improve
- design to include improvements and requirements
- manufacture with improvements
- review new item and improvements
- test of the new item

Failure testing

Valuable processes to perform on a whole consumer product is failure testing or stress testing. In mechanical terms this is the operation of a product until it fails, often under stresses such as increasing vibration, temperature, and humidity. This exposes many unanticipated weaknesses in a product, and the data are used to drive engineering and manufacturing process improvements. Often quite simple changes can dramatically improve product service, such as changing to mold-resistant paint or adding lock-washer placement to the training for new assembly personnel.

Statistical control

Many organizations use statistical process control to bring the organization to Six Sigma levels of quality, in other words, so that the likelihood of an unexpected failure is confined to six standard deviations on the normal distribution. This probability is less than four one-millionths. Items controlled often include clerical tasks such as order-entry as well as conventional manufacturing tasks.

Traditional statistical process controls in manufacturing operations usually proceed by randomly sampling and testing a fraction of the output. Variances in critical tolerances are continuously tracked and where necessary corrected before bad parts are produced.

Total quality management

The quality of products is dependent upon that of the participating constituents, some of which are sustainable and effectively controlled while others are not. The process(es) which are managed with QA pertain to Total Quality Management.

If the specification does not reflect the true quality requirements, the product's quality cannot be guaranteed. For instance, the parameters for a pressure vessel should cover not only the material and dimensions but operating, environmental, safety, reliability and maintainability requirements.

QA in software development

The following are examples of QA models relating to the software development process.

Models and standards

ISO 17025 is an international standard that specifies the general requirements for the competence to carry out tests and or calibrations. There are 15 management requirements and 10 technical requirements. These requirements outline what a laboratory must do to become accredited. Management system refers to the organization's structure for managing its processes or activities that transform inputs of resources into a product or service which meets the organization's objectives, such as satisfying the customer's quality requirements, complying with regulations, or meeting environmental objectives.

The CMMI (Capability Maturity Model Integration) model is widely used to implement Quality Assurance (PPQA) in an organization. The CMMI maturity levels can be divided in to 5 steps, which a company can achieve by performing specific activities within the organization. (CMMI QA processes are excellent for companies like NASA, and may even be adapted for agile development style).

Company quality

During the 1980s, the concept of "company quality" with the focus on management and people came to the fore. It was realized that, if all departments approached quality with an open mind, success was possible if the management led the quality improvement process.

The company-wide quality approach places an emphasis on four aspects :-

1. Elements such as controls, job management, adequate processes, performance and integrity criteria and identification of records
2. Competence such as knowledge, skills, experience, qualifications
3. Soft elements, such as personnel integrity, confidence, organizational culture, motivation, team spirit and quality relationships.
4. Infrastructure (as it enhances or limits functionality)

The quality of the outputs is at risk if any of these aspects is deficient.

QA is not limited to the manufacturing, and can be applied to any business or non-business activity:

- Design work
- Administrative services
- Consulting
- Banking
- Insurance
- Computer software development
- Retailing
- Transportation
- Education

- Translation

It comprises a quality improvement process, which is generic in the sense it can be applied to any of these activities and it establishes a behavior pattern, which supports the achievement of quality.

This in turn is supported by quality management practices which can include a number of business systems and which are usually specific to the activities of the business unit concerned.

In manufacturing and construction activities, these business practices can be equated to the models for quality assurance defined by the International Standards contained in the ISO 9000 series and the specified Specifications for quality systems.

In the system of Company Quality, the work being carried out was shop floor inspection which did not reveal the major quality problems. This led to quality assurance or total quality control, which has come into being recently.

Using contractors and/or consultants

Consultants and contractors are sometimes employed when introducing new quality practices and methods, particularly where the relevant skills and expertise are not available within the organization or when allocating the available internal resources are not available. Consultants and contractors will often employ Quality Management Systems (QMS), auditing and procedural documentation writing CMMI, Six Sigma, Measurement Systems Analysis (MSA), Quality Function Deployment (QFD), Failure Mode and Effects Analysis (FMEA), and Advance Product Quality Planning (APQP).

Quality assurance in European vocational education & training

With the formulation of a joint quality strategy, the European Union seeks to fostering the overall attractiveness of vocational education & training (VET) in Europe. In order to promote this process, a set of new policy instruments were implemented, such as **CQAF** (Common Quality Assurance Framework) and **EQARF** (European Quality Assurance Reference framework), each of which shall contribute to the establishment of a common quality assurance policy and quality culture in VET throughout Europe. Furthermore the new policy instruments shall allow for an increased transparency and mutual trust between national VET systems.

In line with the European quality strategy, the member states subsequently have implemented national structures (QANRPs: reference points for quality assurance in VET), who closely collaborate with national stakeholders in order to meet the requirements and priorities of the national VET systems and support activities to training providers in order to guarantee the implementation and commitment at all levels. At European level, the cooperation between QANRPs will be ensured through the **EQAVET** network.

Over the past few years, with financial support of the European Union as well as the EU member states, numerous pilot initiatives have been developed, most of which are concerned with the promotion and development of quality in VET throughout Europe. Examples can be found in the project database **ADAM**, which keeps comprehensive information about innovation & transfer projects sponsored by the EU.

A practical example might be seen in the **BEQUAL** project, which has developed a benchmarking tool for training providers, who with the help of the online-tool can benchmark their quality performance in line with the CQAF quality process model. Furthermore the project offers a database with European good practice on quality assurance in the field of vocational education & training.

- Online Benchmarking Tool For Vocational Training Institutes
- European Quality Assurance in Vocational Education & Training

Chapter-2

Load Testing

Load testing is the process of putting demand on a system or device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When the load placed on the system is raised beyond normal usage patterns, in order to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so great that error conditions are the expected result, although no clear boundary exists when an activity ceases to be a load test and becomes a stress test.

There is little agreement on what the specific goals of load testing are. The term is often used synonymously with software performance testing, reliability testing, and volume testing. *Load testing* is a type of non-functional testing.

Software load testing

The term *load testing* is used in different ways in the professional software testing community. *Load testing* generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems; often one built using a client/server model, such as web servers. However, other types of software systems can also be load tested. For example, a word processor or graphics editor can be forced to read an extremely large document; or a financial package can be forced to generate a report based on several years' worth of data. The most accurate load testing simulates actual use, as opposed to testing using theoretical or analytical modeling.

Load and performance testing analyzes software intended for a multi-user audience by subjecting the software to different amounts of virtual and live users while monitoring performance measurements under these different loads. Load and performance testing is usually conducted in a test environment identical to the production environment before the software system is permitted to go live.

As an example, a web site with shopping cart capability is required to support 100 concurrent users broken out into following activities:

- 25 Virtual Users (VUsers) log in, browse through items and then log off
- 25 VUsers log in, add items to their shopping cart, check out and then log off
- 25 VUsers log in, return items previously purchased and then log off
- 25 VUsers just log in without any subsequent activity

A test analyst can use various load testing tools to create these VUsers and their activities. Once the test has started and reached a steady state, the application is being tested at the100 VUser load as described above. The application's performance can then be monitored and captured.

The specifics of a load test plan or script will generally vary across organizations. For example, in the bulleted list above, the first item could represent 25 VUsers browsing unique items, random items, or a selected set of items depending upon the test plan or script developed. However, all load test plans attempt to simulate system performance across a range of anticipated peak workflows and volumes. The criteria for passing or failing a load test (pass/fail criteria) are generally different across organizations as well. There are no standards specifying acceptable load testing performance metrics.

A common misconception is that load testing software provides record and playback capabilities like regression testing tools. Load testing tools analyze the entire OSI protocol stack whereas most regression testing tools focus on GUI performance. For example, a regression testing tool will record and playback a mouse click on a button on a web browser, but a load testing tool will send out hypertext the web browser sends after the user clicks the button. In a multiple-user environment, load testing tools can send out hypertext for multiple users with each user having a unique login ID, password, etc.

The popular load testing tools available also provide insight into the causes for slow performance. There are numerous possible causes for slow system performance, including, but not limited to, the following:

- Application server(s) or software
- Database server(s)
- Network – latency, congestion, etc.
- Client-side processing
- Load balancing between multiple servers

Load testing is especially important if the application, system or service will be subject to a service level agreement or SLA.

Load testing tools

Tool Name	Company Name	Notes
IBM Rational Performance Tester	IBM	Eclipse based large scale performance testing tool primarily used for executing large volume performance tests to measure system response time for server based applications. Licensed.
JMeter	An Apache Jakarta open source project	Java desktop application for load testing and performance measurement.
Load Test (included with Soatest	Parasoft	Performance testing tool that verifies functionality and performance under load. Supports SOAtest tests, JUnits, lightweight socket-based components. Detects concurrency issues.
LoadRunner	HP	Performance testing tool primarily used for executing large numbers of tests (or a large number or virtual users) concurrently. Can be used for unit and integration testing as well. Licensed.
OpenSTA	Open System Testing Architecture	Open source web load/stress testing application, licensed under the Gnu GPL. Utilizes a distributed software architecture based on CORBA. OpenSTA binaries available for Windows.
SilkPerformer	Micro Focus	Performance testing in an open and sharable model which allows realistic load tests for thousands of users running business scenarios across a broad range of enterprise application environments.
SLAMD		Open source, 100% Java web application, scriptable, distributed with Tomcat.
Visual Studio Load Test	Microsoft	Visual Studio includes a load test tool which enables a developer to execute a variety of tests (web, unit etc...) with a combination of configurations to simulate real user load.

Mechanical load testing

The purpose of a mechanical load test is to verify that all the component parts of a structure including materials, base-fixings are fit for task and loading it is designed for.

The *Supply of Machinery (Safety) Regulation 1992 UK* state that load testing is undertaken before the equipment is put into service for the first time.

Load testing can be either **Performance**,**Static** or **Dynamic**.

Performance testing is when the stated safe working load (SWL) for a configuration is used to determine that the item performs to the manufactures specification. If an item fails this test then any further tests are pointless.

Static testing is when a load at a factor above the SWL is applied. The item is not operated through all configurations as it is not a requirement of this test.

Dynamic testing is when a load at a factor above the SWL is applied. The item is then operated fully through all configurations and motions. Care must be taken during this test as there is a great risk of catastrophic failure if incorrectly carried out.

The design criteria, relevant legislation or the *Competent Person* will dictate what test is required.

Under the *Lifting Operations and Lifting Equipment Regulations 1998 UK* load testing after the initial test is required if a major component is replaced, if the item is moved from one location to another or as dictated by the *Competent Person*

The loads required for a test are stipulated by the item under test, but here are a few to be aware off. Powered lifting equipment **Static** test to 1.25 SWL and **dynamic** test to 1.1 SWL. Manual lifting equipment **Static** test to 1.5 SWL

For lifting accessories. 2 SWL for items up to 30 tonne capacity. 1.5 SWL for items above 30 tonne capacity. 1 SWL for items above 100 tonnes.

Car charging system

A load test can be used to evaluate the health of a car's battery. The tester consists of a large resistor that has a resistance similar to a car's starter motor and a meter to read the battery's output voltage both in the unloaded and loaded state. When the tester is used, the battery's open circuit voltage is checked first. If the open circuit voltage is below spec (12.6 volts for a fully charged battery), the battery is charged first. After reading the battery's open circuit voltage, the load is applied. When applied, it draws approximately the same current the car's starter motor would draw during cranking. Based on the specified cold cranking amperes of the battery, if the voltage under load falls below a certain point, the battery is bad. Load tests are also used on running cars to check the output of the car's alternator.

Chapter-3

Trent Accreditation Scheme

The **Trent Accreditation Scheme** (TAS), now replaced *de facto* by QHA Trent Accreditation, was a British accreditation scheme formed with a mission to maintain and continually evaluate standards of quality, especially in health care delivery, through the surveying and accreditation of health care organisations, especially hospitals and clinics, both in the UK and elsewhere in the world.

The Trent UK Accreditation Scheme, or TAS UK, ceased to operate in May 2010, when a majority of Board members decided to end the scheme.

Subsequently, the British-based scheme QHA Trent Accreditation began operating.

History of the scheme

The logo of the former Trent Accreditation Scheme

Trent's basic mission resembled that of the USA's Joint Commission International, or JCI, and other major international healthcare accreditation groups, although there were some significant differences in the way the different groups work.

Apart from hospitals in the United Kingdom, Trent also surveyed a large number of private sector hospitals in Hong Kong and at the time of its demise had been developing links with hospitals in Cyprus.

The approach Trent took to Clinic and Hospital Accreditation was based on the axiom that no single healthcare system, whether European, American, Asian or otherwise in origin, has the right to claim a monopoly viewpoint over what represents acceptable quality and best clinical practice throughout the world, and no one country has the absolute right to tell another how their hospitals should be run. What is vital is that the quality of care which patients receive should be of the highest possible standard, and also that the hospitals and clinics providing that care should be independently capable when it comes to working out how best to maintain those standards and how best to respond to any new challenges which will inevitably come along. If the overall standards of a hospital or clinic can be shown to be of acceptable quality, then it is desirable, and even ideal, that local differences related to culture and to legislation should be specifically discussed and incorporated into the assessment standards in an appropriate fashion. That said, Trent was very interested in the medical ethical standards of the hospitals it worked with.

To achieve all of this, Trent worked in close partnership with participating hospitals and clinics to generate an appropriate and mutually acceptable set of standards to survey against. Because the world of healthcare is constantly changing, the standards were constantly reviewed and up-dated through a system of working jointly with representatives of partner hospitals.

Trent developed various ways to ensure local participation, and even ownership, over the accreditation process in a locality. Trent utilised UK-sourced surveyors who were either working in the British National Health Service, or NHS, or had retired in recent times, and hence have valuable experience and insight "at the coal face", and in Hong Kong Trent also appoints locally-domiciled surveyors. Trent surveyors are drawn from a wide variety of professional backgrounds, but especially from the worlds of medicine, dentistry, nursing, the professions complementary to medicine (e.g. physiotherapy, pharmacy etc.) and healthcare management/administration, so as to ensure an appropriately broad portfolio of knowledge and skills are always present within the surveying teams and the wider organisation. Surveyors were all volunteer professionals rather than salaried employees.

Trent surveys were not just a matter of working through a “tick-list” of standards, a process which Trent believed might elevate standards to a certain level but nevertheless do little to inculcate a culture of “thinking for oneself” – instead, Trent surveys involved direct face-to-face conversation with all levels of staff, including clinical medical staff and senior management (for this reason, qualified medical doctors are included in all surveying teams organised by Trent) and Trent surveyors expected full freedom to go anywhere in the hospital or clinic under survey and to talk to anyone they choose to. Discussion and analysis of the data thus generated, not only by the Trent team but also by the hospital or clinic under survey, represented a major component of Trent's approach to hospital and clinic accreditation, and reflected an underlying philosophy that the whole process was about improving services to patients and the ability of an organisation to work effectively towards that aim.

Trent surveyors evaluated a vast range of modalities of a hospital's (or clinic's) activities and governance, including management, estates, equipment, clinical audit, research, education and training, as well as clinical/medical activity. In Hong Kong hospitals, survey teams always consisted of 2 or 3 surveyors from the UK working together with (usually) 2 based in Hong Kong and who were actively working in the local hospitals. One surveyor will be nominated as the lead. The Hong Kong-based surveyors were nominated by the participating hospitals, and after receiving training they always surveyed hospitals other than their own. This approach led to unrivalled opportunity and potential for the sharing of ideas about best practice between hospitals working in the same locality, and the development of camaraderie. Also, patients were spoken to, and their views and experiences also sought.

At the end of a survey, the key findings were initially presented by the Lead Surveyor to the hospital or clinic undergoing the survey, this event taking place almost always on the last day. The findings were subsequently digested, analysed and put into a more detailed printed report, with positive virtues being highlighted as well as problems. However, because of the end-of-survey oral presentation, hospitals and clinics could start putting remedial action into place as soon as possible.

After a round of surveys, a joint meeting was held at which the printed reports of all the hospital and clinic surveys conducted in that particular round are discussed jointly and in depth by the Trent Board (which had both local and UK representation) together with senior representatives of the hospital or clinic being surveyed, and a decision was then taken as to whether or not accreditation would be granted unconditionally, or if it would be subject to conditions.

The Trent approach to accreditation ensured that the local hospitals and clinics enjoyed some ownership over the whole process, which would not be the case if all of the standards, all of the surveyors and all of the decisions regarding who was successful or not in achieving accreditation were imposed unilaterally from outside. It helped to build up the confidence of participating hospitals in their ability to develop ways to maintain and improve quality in a way that schemes which operate da more didactic approach to standards and their assessment would not. It also meant that there were Trent surveyors constantly present in the majority of the scheme's participating hospitals.

Trent was a member of the United Kingdom Accreditation Forum (UKAF) and an institutional member of ISQUA.

International Healthcare Accreditation

With the advent of medical tourism, international healthcare accreditation has increasingly grown in importance. A number of accreditation organisations sourced from a number of countries fulfil this internationally-orientated role, including:

- Accreditation Canada International (formerly "The Canadian Council on Health Services Accreditation", or CCHSA)
- Joint Commission International (JCI), in the USA
- The Australian Council on Healthcare Standards, or ACHS
- QHA Trent Accreditation (UK)

No single accreditation scheme enjoys exclusive rights to be seen as an overall world-wide-relevant scheme, and some hospitals are looking towards multiple accreditation to achieve performance credibility in different parts of the world.

The Trent Scheme was the first accreditation scheme to survey and accredit a hospital in Asia, in Hong Kong in 2000 . Since then others such as JCI have entered the market, with JCI first accrediting Bumrungrad International Hospital in Thailand in 2002.

Closure of Trent Accreditation Scheme

The UK-based Trent Accreditation Scheme (TAS UK) ceased surveying and accreditation activities in 2010. Subsequently, the British-based scheme QHA Trent Accreditation began operating.

Chapter-4

QCReporting and Software Assurance

QCReporting

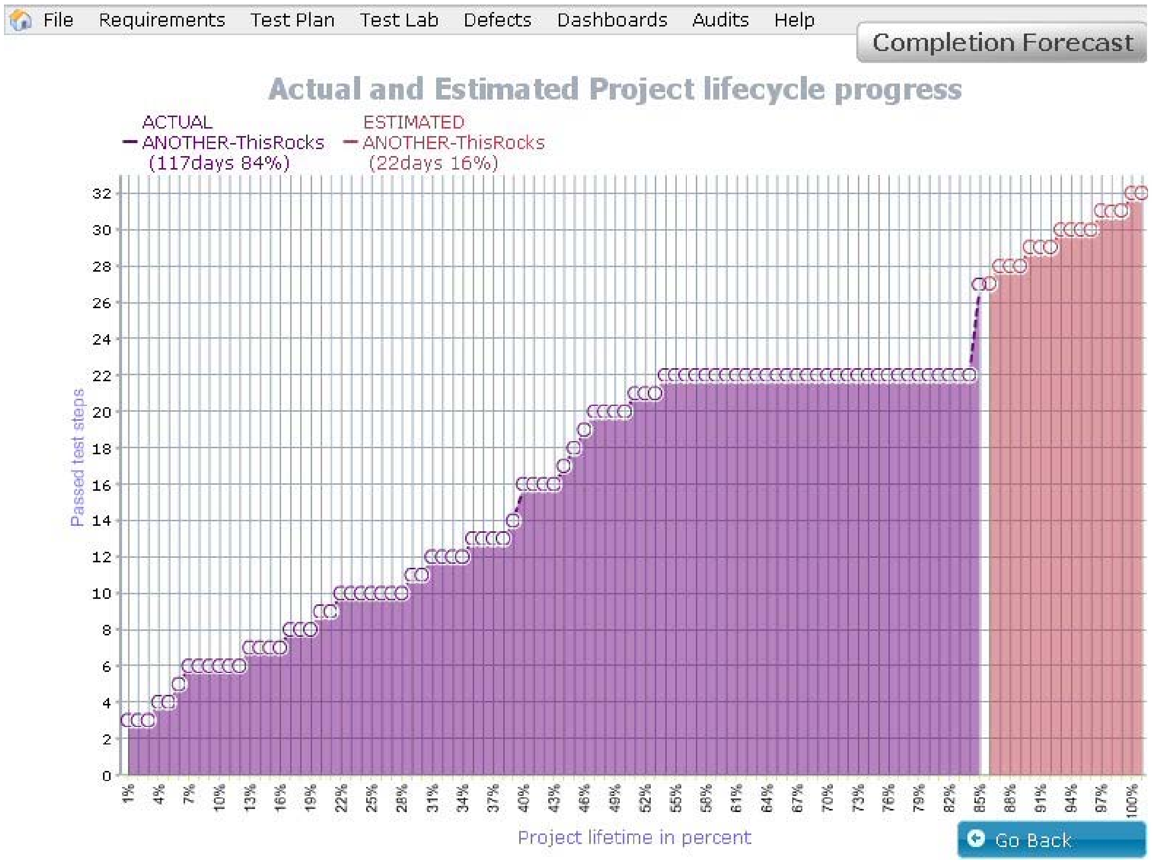
QC Reporting is a web based reporting tool that extracts data to produce reports from the database used by the test tool HP Quality Center and its predecessor HP Test Director. The resulting reporting media format can be, where possible:

- PDF
- Microsoft Excel
- Screen table
- Adobe Flash Graph
- Microsoft Word
- HTML

This product was formerly known as *TD Reports* which was based on the predecessor to Quality Center known as Test director during the former Mercury Interactive days.

QCReporting benefits testing resources who spending time writing and maintaining test reports by enabling them to produce reports quickly and accurately. QCreporting is usually installed as part of a company's intranet. This means that reports can be created and presented easily during meetings using the company Wifi whilst containing up-to-date data.

Need for some QC reporting tool



QCReporting Completion forecast

The testing product that is used with QCReporting is Quality Center. Quality Center comes with some reports. These reports can be built and set by the user but they often lead to very long HTML output. Over the years testers, usually the *test manager* of a test team would spend time creating and formatting reports for senior management. This process has been known to take a full working day to compile, usually involving multiple Excel report extracts. This in turn led to the writing of custom reports. These were often developed using one of two ways.

- By connecting to Quality Center using 'VBScript' report script which extracts the data and creates a formatted report. The format of the report is set in the code.
- By scripting with VBScript inside Quality Center in it's workflow area. Quality Center allows adding custom buttons to its dashboard, which can then be used to trigger the custom report script.

The former option became the most popular with extracts also being written to Microsoft Excel which could then be used to create graphs.

The need for up-to-date information spurred on ever more ways to extract data from Quality Center and format a report.

Example of a VB Script used to extract a QC report

The following VBScript can be executed by typing **wscript scriptname.vbs** at the command prompt.

The tool helps make the need for scripts such as these a thing of the past. The following script is an example of what a test manager would have had to use to extract data from Quality Center.

```
Option Explicit
Dim QCConnection
'--------
Dim Excel, Sheet, CurrRow
Dim UserN, QCUserN, UserP, QCDomain, QCProj, QCAddr, XLSSavePos
Dim REQUsersTreeStr, WSHNetwork
'--------
QCUserN  = "QC_User_Name"
UserP    = "QC_User_Password"
QCDomain = "QC_Domain_Name"
QCProj   = "QC_Project_Name"
QCAddr   = "http://SomeServer:8080/qcbin"
'--------
WScript.Echo "Initialising...."
Set Excel = CreateObject("Excel.Application") 'Open Excel
Excel.ScreenUpdating = False
Set WSHNetwork  = WScript.CreateObject("WScript.Network")
UserN = WshNetwork.UserName
Set WSHNetwork = Nothing
Set QCConnection = CreateObject("TDApiOle80.TDConnection")
QCConnection.InitConnectionEx QCAddr
QCConnection.login QCUserN, UserP
QCConnection.Connect QCDomain, QCProj
If (QCConnection.Connected <> True) Then
    WScript.Echo "QC Project Failed to Connect"
    WScript.Quit
End If
'--------
On Error Resume Next
WScript.Echo "Extract from QC here and format it out"
'--------
If Err.number <> 0 Then WScript.Echo Err.Description & ":" & Err.Number
'--------
'Note: This will attempt to save the XLS into UserN's DESKTOP. please
check the folder location first
XLSSavePos = "C:\Documents and Settings\" & UserN & "\Desktop\" &
QCProj & ".xls"
Excel.ScreenUpdating = True
WScript.Echo "Attempting to save to folder:'" & XLSSavePos & "'"
Excel.ActiveWorkbook.SaveAs(XLSSavePos)
Excel.Quit
Set Excel = Nothing
'--------
QCConnection.Disconnect
QCConnection.Logout
QCConnection.ReleaseConnection
WScript.Echo "Report Complete"
```

Marriage of QCReporting to Quality Center

There are a number of ways where a marriage relationship is apparent between the reporting tool and Quality Center.

- Users are not created in QCReporting, instead they are synchronised from Quality Center. The syncyhronisation includes their user settings and access rights to domains and projects.
- Data managed and altered in Quality Center is reported on using a read-only user in QCReporting.
- Customisation, such as user fields, made in Quality Center are reported on through QCReporting.

QCReporting tool support

The tool offers user support and functionality to manage itself.

User Support

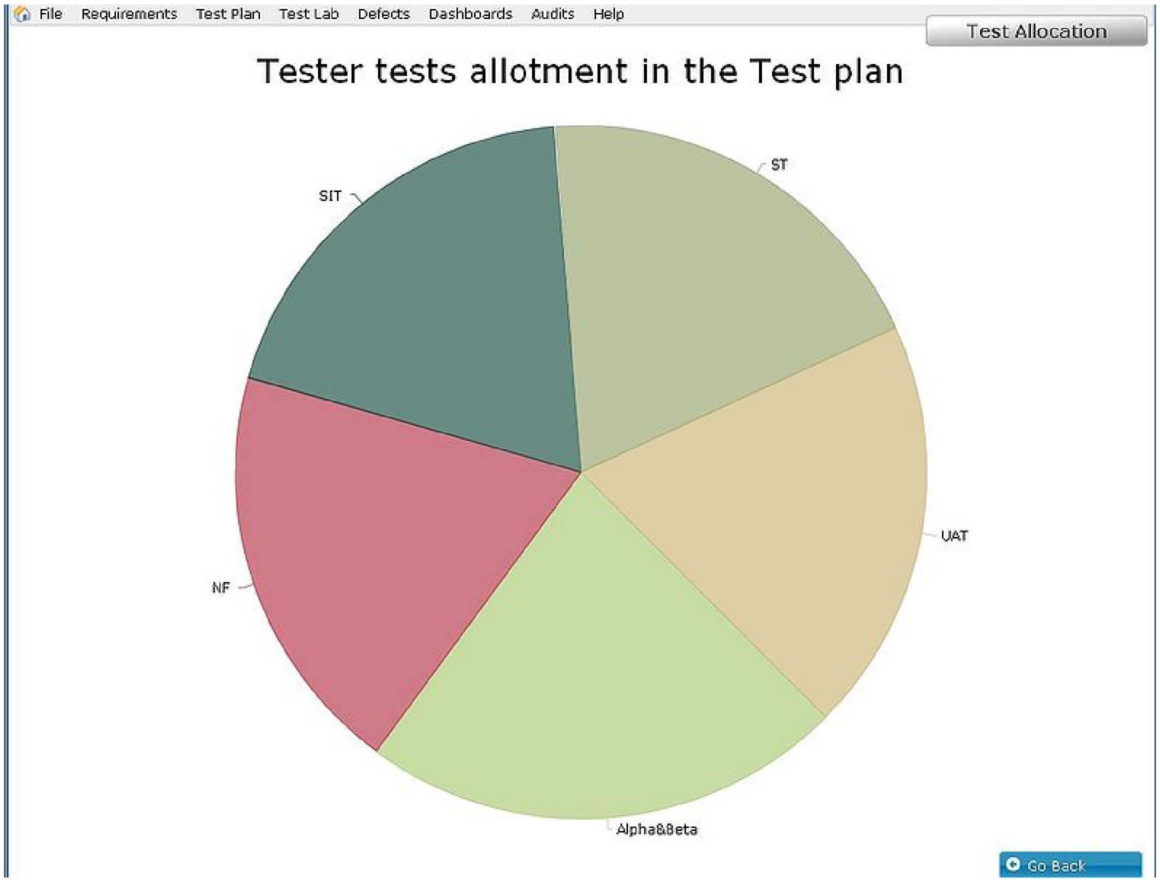
User support is primarily managed through a web based ticket system off the clients own installed instance of the tool.

- Tickets can be user or group owned
- Tickets can be escalated
- Simple requests such as wanting a new report can be raised as a ticket
- Fixes to tickets appear through the automatic update system

Tool maintenance

- An **auto update** facility downloads and installs the latest updates from the support site. The update is encrypted over https.
- The tool verifies each transmission with its unique client license information

QCReporting was born



QCReporting Test Allocation

Quality Center is a testing tool that is very common in the testing world. The test tool is divided into four main categories. The QCReporting follows those categories. They are:

- Requirements & coverage assurance report. These can be business, design, functional and non-functional requirements.
- Test plan
- Test Lab
- Defects
- Dashboards. These are overview cross-project and cross-domain reports. Most popular with upper management.
- Test Exit reports - at the end of every phase and cycle issue the report to summarise the finding, the coverage and the outcome.
- The additional category is Audit reports which follow the Sarbanes–Oxley legislation.

The tool delivers fantastic time saving and greater levels of accuracy: producing the Test Exit report, which includes all the customised fields of QC is just one button click away; or a few if you would like to exclude certain parts of the default settings.

The tool provides comprehensive overview of the test coverage reports showing horizontal and vertical traceability between all of the project requirements and test scripts whilst highlighting where there may be gaps, such as missing requirements or orphaned test scripts.

"How are we doing?" Many Project and Test Managers have heard this never-aging question. How do you explain it in not so many words, as there is little patience, time or desire to listen to a lengthy and wordy explanation. With QCReporting the answer is just one click away - a comprehensive and yet easy-to-understand dashboard spread across several projects and even domains can appear within mere seconds. It tells you where you are with each project and across the function or the domain and provides some vital indicators of the project "health" and its testing phase status and maturity levels.

The tool plays a vital role in the Impact Assessment procedure, providing conscious and unambiguous views of the real impact of the change or of the delay to the defect fix. There is no longer a need to guess - one click away is the information on which requirements were affected and to which extent, which scripts need to be re-writted and which ones will have to be amended; what the regression implication is and, if related to the already tested piece of work, how long did it take to execute in the past.

There are several additional features which will be available shortly.

What's even better is that one of the tool's add-ins such as the Auto-Updater is used for its self-maintenance and to provide instant user support.

Software assurance

Software Assurance (SwA) is defined as "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the software functions in the intended manner."

Alternate definitions

Department of Homeland Security (DHS)

According to the DHS, software assurance addresses:

- Trustworthiness - No exploitable vulnerabilities exist, either maliciously or unintentionally inserted;
- Predictable Execution - Justifiable confidence that software, when executed, functions as intended;
- Conformance - Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards/ procedures.

Contributing SwA disciplines, articulated in Bodies of Knowledge and Core Competencies: Software Engineering, Systems Engineering, Information Systems Security Engineering, Information Assurance, Test and Evaluation, Safety, Security, Project Management, and Software Acquisition.

Software Assurance is a strategic initiative of the U.S. Department of Homeland Security (DHS) to promote integrity, security, and reliability in software. The SwA Program is based upon the National Strategy to Secure Cyberspace - Action/Recommendation 2-14:

“DHS will facilitate a national public-private effort to promulgate best practices and methodologies that promote integrity, security, and reliability in software code development, including processes and procedures that diminish the possibilities of erroneous code, malicious code, or trap doors that could be introduced during development.”

United States Department of Defense (DoD)

According to the DoD, software assurance relates to "the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software."

Software Assurance Metrics and Tool Evaluation (SAMATE) project

According to the NIST SAMATE project, software assurance is "the planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures to help achieve:

- Trustworthiness - No exploitable vulnerabilities exist, either of malicious or unintentional origin, and
- Predictable Execution - Justifiable confidence that software, when executed, functions as intended."

National Aeronautics and Space Administration (NASA)

According to NASA, Software Assurance is a "planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures. It includes the disciplines of Quality Assurance, Quality Engineering, Verification and Validation, Nonconformance Reporting and Corrective Action, Safety Assurance, and Security Assurance and their application during a software life cycle." The NASA Software Assurance Standard also states: "The application of these disciplines during a software development life cycle is called Software Assurance."

Object Management Group (OMG)

According to the OMG, Software Assurance is “justifiable trustworthiness in meeting established business and security objectives.”

OMG's SwA Special Interest Group (SIG), works with Platform and Domain Task Forces and other software industry entities and groups external to the OMG, to coordinate the establishment of a common framework for analysis and exchange of information related to software trustworthiness by facilitating the development of a specification for a Software Assurance Framework that will:

- Establish a common framework of software properties that can be used to represent any/all classes of software so software suppliers and acquirers can represent their claims and arguments(respectively), along with the corresponding evidence, employing automated tools (to address scale)
- Verify that products have sufficiently satisfied these characteristics in advance of product acquisition, so that system engineers/integrators can use these products to build (compose) larger assured systems with them
- Enable industry to improve visibility into the current status of software assurance during development of its software
- Enable industry to develop automated tools that support the common framework.

Software Assurance Forum for Excellence in Code (SAFECode)

According to SAFECode, Software Assurance is “confidence that software, hardware and services are free from intentional and unintentional vulnerabilities and that the software functions as intended.”

Webopedia

According to Webopedia, Software Quality Assurance, abbreviated as SQA, and also called "software assurance", is a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or inserted at anytime during its lifecycle, and that the software functions in the intended manner."

As indicated in the Webopedia definition, the term "software assurance" has been used as a shorthand for Software Quality Assurance (SQA) when not necessarily considering security or trustworthiness. SQA is defined in the *Handbook of Software Quality Assurance* as: "the set of systematic activities providing evidence of the ability of the software process to produce a software product that is fit to use."

Chapter-5

Software Quality

In the context of software engineering, **software quality** measures how well software is designed (*quality of design*), and how well the software conforms to that design (*quality of conformance*), although there are several different definitions, including conformance to customer expectectations. It is often described as the 'fitness for purpose' of a piece of software.

Whereas *quality of conformance* is concerned with implementation, *quality of design* measures how valid the design and requirements are in creating a worthwhile product.

Definition

One of the challenges of software quality is that "everyone feels they understand it".

In addition to more software specific definitions given below, there are several applicable definitions of quality which are used in business.Quality_(business)#Definitions

Software quality may be defined as conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software.

The three key points in this definition:

1. Software requirements are the foundations from which quality is measured.

   Lack of conformance to requirement is lack of quality.

2. Specified standards define a set of development criteria that guide the manager is software engineering.

   If criteria are not followed lack of quality will usually result.

3. A set of implicit requirements often goes unmentioned, for example ease of use, maintainability etc.

   If software confirms to its explicit requirement but fails to meet implicit requirements, software quality is suspected.

A definition in Steve McConnell's *Code Complete* divides software into two pieces: **internal** and **external quality characteristics**. External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not.

Another definition by Dr. Tom DeMarco says "a product's quality is a function of how much it changes the world for the better." This can be interpreted as meaning that user satisfaction is more important than anything in determining software quality.

Another definition, coined by Gerald Weinberg in *Quality Software Management: Systems Thinking*, is "Quality is value to some person." This definition stresses that quality is inherently subjective - different people will experience the quality of the same software very differently. One strength of this definition is the questions it invites software teams to consider, such as "Who are the people we want to value our software?" and "What will be valuable to *them*?"

History

Software product quality

- Product quality
    - conformance to requirements or program specification; related to Reliability
- Scalability
- Correctness
- Completeness
- Absence of bugs
- Fault-tolerance
    - Extensibility
    - Maintainability
- Documentation

The Consortium for IT Software Quality (CISQ) was launched in 2009 to standardize the measurement of software product quality. The Consortium's goal is to bring together industry executives from Global 2000 IT organizations, system integrators, outsourcers, and package vendors to jointly address the challenge of standardizing the measurement of IT software quality and to promote a market-based ecosystem to support its deployment.

Source code quality

A computer has no concept of "well-written" source code. However, from a human point of view source code can be written in a way that has an effect on the effort needed to comprehend its behavior. Many source code programming style guides, which often stress readability and usually language-specific conventions are aimed at reducing the cost of source code maintenance. Some of the issues that affect code quality include:

- Readability
- Ease of maintenance, testing, debugging, fixing, modification and portability
- Low complexity
- Low resource consumption: memory, CPU
- Number of compilation or lint warnings
- Robust input validation and error handling, established by software fault injection

Methods to improve the quality:

- Refactoring
- Code Inspection or software review
- Documenting code

Software reliability

Software **reliability** is an important facet of software quality. It is defined as "the probability of failure-free operation of a computer program in a specified environment for a specified time".

One of reliability's distinguishing characteristics is that it is objective, measurable, and can be estimated, whereas much of software quality is subjective criteria. This distinction is especially important in the discipline of Software Quality Assurance. These measured criteria are typically called software metrics.

History

With software embedded into many devices today, software failure has caused more than inconvenience. Software errors have even caused human fatalities. The causes have ranged from poorly designed user interfaces to direct programming errors. An example of a programming error that lead to multiple deaths is discussed in Dr. Leveson's paper (PDF). This has resulted in requirements for development of some types software. In the United States, both the Food and Drug Administration (FDA) and Federal Aviation Administration (FAA) have requirements for software development.

Goal of reliability

The need for a means to objectively determine software reliability comes from the desire to apply the techniques of contemporary engineering fields to the development of software. That desire is a result of the common observation, by both lay-persons and specialists, that computer software does not work the way it ought to. In other words, software is seen to exhibit undesirable behaviour, up to and including outright failure, with consequences for the data which is processed, the machinery on which the software runs, and by extension the people and materials which those machines might negatively affect. The more critical the application of the software to economic and production processes, or to life-sustaining systems, the more important is the need to assess the software's reliability.

Regardless of the criticality of any single software application, it is also more and more frequently observed that software has penetrated deeply into most every aspect of modern life through the technology we use. It is only expected that this infiltration will continue, along with an accompanying dependency on the software by the systems which maintain our society. As software becomes more and more crucial to the operation of the systems on which we depend, the argument goes, it only follows that the software should offer a concomitant level of dependability. In other words, the software should behave in the way it is intended, or even better, in the way it should.

Challenge of reliability

The circular logic of the preceding sentence is not accidental—it is meant to illustrate a fundamental problem in the issue of measuring software reliability, which is the difficulty of determining, in advance, exactly how the software is intended to operate. The problem seems to stem from a common conceptual error in the consideration of software, which is that software in some sense takes on a role which would otherwise be filled by a human being. This is a problem on two levels. Firstly, most modern software performs work which a human could never perform, especially at the high level of reliability that is often expected from software in comparison to humans. Secondly, software is fundamentally incapable of most of the mental capabilities of humans which separate them from mere mechanisms: qualities such as adaptability, general-purpose knowledge, a sense of conceptual and functional context, and common sense.

Nevertheless, most software programs could safely be considered to have a particular, even singular purpose. If the possibility can be allowed that said purpose can be well or even completely defined, it should present a means for at least considering objectively whether the software is, in fact, reliable, by comparing the expected outcome to the actual outcome of running the software in a given environment, with given data. Unfortunately, it is still not known whether it is possible to exhaustively determine either the expected outcome or the actual outcome of the entire set of possible environment and input data to a given program, without which it is probably impossible to determine the program's reliability with any certainty.

However, various attempts are in the works to attempt to rein in the vastness of the space of software's environmental and input variables, both for actual programs and theoretical descriptions of programs. Such attempts to improve software reliability can be applied at different stages of a program's development, in the case of real software. These stages principally include: requirements, design, programming, testing, and runtime evaluation. The study of theoretical software reliability is predominantly concerned with the concept of correctness, a mathematical field of computer science which is an outgrowth of language and automata theory.

Reliability in program development

Requirements

A program cannot be expected to work as desired if the developers of the program do not, in fact, know the program's desired behaviour in advance, or if they cannot at least determine its desired behaviour in parallel with development, in sufficient detail. What level of detail is considered sufficient is hotly debated. The idea of perfect detail is attractive, but may be impractical, if not actually impossible. This is because the desired behaviour tends to change as the possible range of the behaviour is determined through actual attempts, or more accurately, failed attempts, to achieve it.

Whether a program's desired behaviour can be successfully specified in advance is a moot point if the behaviour cannot be specified at all, and this is the focus of attempts to formalize the process of creating requirements for new software projects. In situ with the formalization effort is an attempt to help inform non-specialists, particularly non-programmers, who commission software projects without sufficient knowledge of what computer software is in fact capable. Communicating this knowledge is made more difficult by the fact that, as hinted above, even programmers cannot always know in advance what is actually possible for software in advance of trying.

Design

While requirements are meant to specify what a program should do, design is meant, at least at a high level, to specify how the program should do it. The usefulness of design is also questioned by some, but those who look to formalize the process of ensuring reliability often offer good software design processes as the most significant means to accomplish it. Software design usually involves the use of more abstract and general means of specifying the parts of the software and what they do. As such, it can be seen as a way to break a large program down into many smaller programs, such that those smaller pieces together do the work of the whole program.

The purposes of high-level design are as follows. It separates what are considered to be problems of architecture, or overall program concept and structure, from problems of actual coding, which solve problems of actual data processing. It applies additional constraints to the development process by narrowing the scope of the smaller software components, and thereby—it is hoped—removing variables which could increase the likelihood of programming errors. It provides a program template, including the specification of interfaces, which can be shared by different teams of developers working on disparate parts, such that they can know in advance how each of their contributions will interface with those of the other teams. Finally, and perhaps most controversially, it specifies the program independently of the implementation language or languages, thereby removing language-specific biases and limitations which would otherwise creep into the design, perhaps unwittingly on the part of programmer-designers.

Programming

The history of computer programming language development can often be best understood in the light of attempts to master the complexity of computer programs, which otherwise becomes more difficult to understand in proportion (perhaps exponentially) to the size of the programs. (Another way of looking at the evolution of programming languages is simply as a way of getting the computer to do more and more of the work, but this may be a different way of saying the same thing). Lack of understanding of a program's overall structure and functionality is a sure way to fail to detect errors in the program, and thus the use of better languages should, conversely, reduce the number of errors by enabling a better understanding.

Improvements in languages tend to provide incrementally what software design has attempted to do in one fell swoop: consider the software at ever greater levels of abstraction. Such inventions as statement, sub-routine, file, class, template, library, component and more have allowed the arrangement of a program's parts to be specified using abstractions such as layers, hierarchies and modules, which provide structure at different granularities, so that from any point of view the program's code can be imagined to be orderly and comprehensible.

In addition, improvements in languages have enabled more exact control over the shape and use of data elements, culminating in the abstract data type. These data types can be specified to a very fine degree, including how and when they are accessed, and even the state of the data before and after it is accessed.

Software Build and Deployment

Many programming languages such as C and Java require the program "source code" to be translated in to a form that can be executed by a computer. This translation is done by a program called a compiler. Additional operations may be involved to associate, bind, link or package files together in order to create a usable runtime configuration of the software application. The totality of the compiling and assembly process is generically called "building" the software.

The software build is critical to software quality because if any of the generated files are incorrect the software build is likely to fail. And, if the incorrect version of a program is inadvertently used, then testing can lead to false results.

Software builds are typically done in work area unrelated to the runtime area, such as the application server. For this reason, a deployment step is needed to physically transfer the software build products to the runtime area. The deployment procedure may also involve technical parameters, which, if set incorrectly, can also prevent software testing from beginning. For example, a Java application server may have options for parent-first or parent-last class loading. Using the incorrect parameter can cause the application to fail to execute on the application server.

The technical activities supporting software quality including build, deployment, change control and reporting are collectively known as Software configuration management. A number of software tools have arisen to help meet the challenges of configuration management including file control tools and build control tools.

Testing

Software testing, when done correctly, can increase overall software *quality of conformance* by testing that the product conforms to its requirements. Testing includes, but is not limited to:

1. Unit Testing
2. Functional Testing
3. Regression Testing
4. Performance Testing
5. Failover Testing
6. Usability Testing

A number of agile methodologies use testing early in the development cycle to ensure quality in their products. For example, the test-driven development practice, where tests are written before the code they will test, is used in Extreme Programming to ensure quality.

Runtime

runtime reliability determinations are similar to tests, but go beyond simple confirmation of behaviour to the evaluation of qualities such as performance and interoperability with other code or particular hardware configurations.

Software quality factors

A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program. Note that none of these factors are binary; that is, they are not “either you have it or you don’t” traits. Rather, they are characteristics that one seeks to maximize in one’s software to optimize its quality. So rather than asking whether a software product “has” factor *x*, ask instead the *degree* to which it does (or does not).

Some software quality factors are listed here:

Understandability

Clarity of purpose. This goes further than just a statement of purpose; all of the design and user documentation must be clearly written so that it is easily understandable. This is obviously subjective in that the user context must be taken into account: for instance, if the software product is to be used by software engineers it is not required to be understandable to the layman.

Completeness

Presence of all constituent parts, with each part fully developed. This means that if the code calls a subroutine from an external library, the software package must provide reference to that library and all required parameters must be passed. All required input data must also be available.

Conciseness

Minimization of excessive or redundant information or processing. This is important where memory capacity is limited, and it is generally considered good practice to keep lines of code to a minimum. It can be improved by replacing repeated functionality by one subroutine or function which achieves that functionality. It also applies to documents.

Portability

Ability to be run well and easily on multiple computer configurations. Portability can mean both between different hardware—such as running on a PC as well as a smartphone—and between different operating systems—such as running on both Mac OS X and GNU/Linux.

Consistency

Uniformity in notation, symbology, appearance, and terminology within itself.

Maintainability

Propensity to facilitate updates to satisfy new requirements. Thus the software product that is maintainable should be well-documented, should not be complex, and should have spare capacity for memory, storage and processor utilization and other resources.

Testability

Disposition to support acceptance criteria and evaluation of performance. Such a characteristic must be built-in during the design phase if the product is to be easily testable; a complex design leads to poor testability.

Usability

Convenience and practicality of use. This is affected by such things as the human-computer interface. The component of the software that has most impact on this is the user interface (UI), which for best usability is usually graphical (i.e. a GUI).

Reliability

Ability to be expected to perform its intended functions satisfactorily. This implies a time factor in that a reliable product is expected to perform correctly over a period of time. It also encompasses environmental considerations in that the product is required to perform correctly in whatever conditions it finds itself (sometimes termed robustness).

Efficiency

Fulfillment of purpose without waste of resources, such as memory, space and processor utilization, network bandwidth, time, etc.

Security

Ability to protect data against unauthorized access and to withstand malicious or inadvertent interference with its operations. Besides the presence of appropriate security mechanisms such as authentication, access control and encryption, security also implies resilience in the face of malicious, intelligent and adaptive attackers.

Measurement of software quality factors

There are varied perspectives within the field on measurement. There are a great many measures that are valued by some professionals—or in some contexts, that are decried as harmful by others. Some believe that quantitative measures of software quality are essential. Others believe that contexts where quantitative measures are useful are quite rare, and so prefer qualitative measures. Several leaders in the field of software testing have written about the difficulty of measuring what we truly want to measure well.

One example of a popular metric is the number of faults encountered in the software. Software that contains few faults is considered by some to have higher quality than software that contains many faults. Questions that can help determine the usefulness of this metric in a particular context include:

1. What constitutes "many faults?" Does this differ depending upon the purpose of the software (e.g., blogging software vs. navigational software)? Does this take into account the size and complexity of the software?
2. Does this account for the importance of the bugs (and the importance to the stakeholders of the people those bugs bug)? Does one try to weight this metric by the severity of the fault, or the incidence of users it affects? If so, how? And if not, how does one know that 100 faults discovered is better than 1000?
3. If the count of faults being discovered is shrinking, how do I know what that means? For example, does that mean that the product is now higher quality than it was before? Or that this is a smaller/less ambitious change than before? Or that fewer tester-hours have gone into the project than before? Or that this project was tested by less skilled testers than before? Or that the team has discovered that fewer faults reported is in their interest?

This last question points to an especially difficult one to manage. All software quality metrics are in some sense measures of human behavior, since humans create software. If a team discovers that they will benefit from a drop in the number of reported bugs, there is a strong tendency for the team to start reporting fewer defects. That may mean that email begins to circumvent the bug tracking system, or that four or five bugs get lumped into one bug report, or that testers learn not to report minor annoyances. The difficulty is measuring what we mean to measure, without creating incentives for software programmers and testers to consciously or unconsciously "game" the measurements.

Software quality factors cannot be measured because of their vague definitions. It is necessary to find measurements, or metrics, which can be used to quantify them as non-functional requirements. For example, reliability is a software quality factor, but cannot be evaluated in its own right. However, there are related attributes to reliability, which can indeed be measured. Some such attributes are mean time to failure, rate of failure occurrence, and availability of the system. Similarly, an attribute of portability is the number of target-dependent statements in a program.

A scheme that could be used for evaluating software quality factors is given below. For every characteristic, there are a set of questions which are relevant to that characteristic. Some type of scoring formula could be developed based on the answers to these questions, from which a measurement of the characteristic can be obtained.

Understandability

Are variable names descriptive of the physical or functional property represented? Do uniquely recognisable functions contain adequate comments so that their purpose is clear? Are deviations from forward logical flow adequately commented? Are all elements of an array functionally related?...

Completeness

Are all necessary components available? Does any process fail for lack of resources or programming? Are all potential pathways through the code accounted for, including proper error handling?

Conciseness

Is all code reachable? Is any code redundant? How many statements within loops could be placed outside the loop, thus reducing computation time? Are branch decisions too complex?

Portability

Does the program depend upon system or library routines unique to a particular installation? Have machine-dependent statements been flagged and commented? Has dependency on internal bit representation of alphanumeric or special characters been avoided? How much effort would be required to transfer the program from one hardware/software system or environment to another?

Consistency

Is one variable name used to represent different logical or physical entities in the program? Does the program contain only one representation for any given physical or mathematical constant? Are functionally similar arithmetic expressions similarly constructed? Is a consistent scheme used for indentation, nomenclature, the color palette, fonts and other visual elements?

Maintainability

Has some memory capacity been reserved for future expansion? Is the design cohesive—i.e., does each module have distinct, recognizable functionality? Does the software allow for a change in data structures (object-oriented designs are more likely to allow for this)? If the code is procedure-based (rather than object-oriented), is a change likely to require restructuring the main program, or just a module?

Testability

Are complex structures employed in the code? Does the detailed design contain clear pseudo-code? Is the pseudo-code at a higher level of abstraction than the code? If tasking is used in concurrent designs, are schemes available for providing adequate test cases?

Usability

Is a GUI used? Is there adequate on-line help? Is a user manual provided? Are meaningful error messages provided?

Reliability

Are loop indexes range-tested? Is input data checked for range errors? Is divide-by-zero avoided? Is exception handling provided? It is the probability that the software performs its intended functions correctly in a specified period of time under stated operation conditions, but there could also be a problem with the requirement document...

Efficiency

Have functions been optimized for speed? Have repeatedly used blocks of code been formed into subroutines? Has the program been checked for memory leaks or overflow errors?

Security

Does the software protect itself and its data against unauthorized access and use? Does it allow its operator to enforce security policies? Are security mechanisms appropriate, adequate and correctly implemented? Can the software withstand attacks that can be anticipated in its intended environment?

User's perspective

In addition to the technical qualities of software, the end user's experience also determines the quality of software. This aspect of software quality is called usability. It is hard to quantify the usability of a given software product. Some important questions to be asked are:

- Is the user interface intuitive (self-explanatory/self-documenting)?
- Is it easy to perform simple operations?
- Is it feasible to perform complex operations?
- Does the software give sensible error messages?
- Do widgets behave as expected?
- Is the software well documented?
- Is the user interface responsive or too slow?

Also, the availability of (free or paid) support may factor into the usability of the software.

Chapter-6

Software Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

Overview

Testing can never completely identify all the defects within software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behavior of the product against oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. **Software testing** is the process of attempting to make this assessment.

A study conducted by NIST in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed.

History

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979. Although his attention was on breakage testing ("a successful test is one that finds a bug") it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing in the following stages:

- Until 1956 - Debugging oriented
- 1957–1978 - Demonstration oriented
- 1979–1982 - Destruction oriented
- 1983–1987 - Evaluation oriented
- 1988–2000 - Prevention oriented

Software testing topics

Scope

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Functional vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirement gaps, e.g., unrecognized requirements, that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

Finding faults early

It is commonly believed that the earlier a defect is found the cheaper it is to fix it. The following table shows the cost of fixing the defect depending on the stage it was found. For example, if a problem in the requirements is found only post-release, then it would cost 10–100 times more to fix than if it had already been found by the requirements review.

Cost to fix a defect		**Time detected**				
		Requirements	**Architecture**	**Construction**	**System test**	**Post-release**
Time introduced	**Requirements**	1×	3×	5–10×	10×	10–100×
	Architecture	-	1×	10×	15×	25–100×

Construction	-	-	1×	10×	10–25×

Compatibility

A common cause of software failure (real or perceived) is a lack of compatibility with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a web application, which must render in a web browser). For example, in the case of a lack of backward compatibility, this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment was capable of using. Sometimes such issues can be fixed by proactively abstracting operating system functionality into a separate program module or library.

Input combinations and preconditions

A very fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*)—usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

Static vs. dynamic testing

There are many approaches to software testing. Reviews, walkthroughs, or inspections are considered as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing can be (and unfortunately in practice often is) omitted. Dynamic testing takes place when the program itself is used for the first time (which is generally considered the beginning of the testing stage). Dynamic testing may begin before the program is 100% complete in order to test particular sections of code (modules or discrete functions). Typical techniques for this are either using stubs/drivers or execution from a debugger environment. For example, spreadsheet programs are, by their very nature, tested to a large extent interactively ("on the fly"), with results displayed immediately after each calculation or text manipulation.

Software verification and validation

Software testing is used in association with verification and validation:

- Verification: Have we built the software right? (i.e., does it match the specification).
- Validation: Have we built the right software? (i.e., is this what the customer wants).

The terms verification and validation are commonly used interchangeably in the industry; it is also common to see these two terms incorrectly defined. According to the IEEE Standard Glossary of Software Engineering Terminology:

> Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
> Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

The software testing team

Software testing can be done by software testers. Until the 1980s the term "software tester" was used generally, but later it was also seen as a separate profession. Regarding the periods and the different goals in software testing, different roles have been established: *manager*, *test lead*, *test designer*, *tester*, *automation developer*, and *test administrator*.

Software quality assurance (SQA)

Though controversial, software testing is a part of the software quality assurance (SQA) process. In SQA, software process specialists and auditors are concerned for the software development process rather than just the artefacts such as documentation, code and systems. They examine and change the software engineering process itself to reduce the amount of faults that end up in the delivered software: the so-called *defect rate.*

What constitutes an "acceptable defect rate" depends on the nature of the software; A flight simulator video game would have much higher defect tolerance than software for an actual airplane.

Although there are close links with SQA, testing departments often exist independently, and there may be no SQA function in some companies.

Software testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, QA (quality assurance) is the implementation of policies and procedures intended to prevent defects from occurring in the first place.

Testing methods

The box approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

White box testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

Types of white box testing

The following types of white box testing exist:

- API testing (application programming interface) - testing of the application using public and private APIs
- Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods - improving the coverage of a test by introducing faults to test code paths
- Mutation testing methods
- Static testing - White box testing includes all static testing

Test coverage

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

Two common forms of code coverage are:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test

They both return a code coverage metric, measured as a percentage.

Black box testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, exploratory testing and specification-based testing.

Specification-based testing: Specification-based testing aims to test the functionality of software according to the applicable requirements. Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Specification-based testing is necessary, but it is insufficient to guard against certain risks.

Advantages and disadvantages: The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code *must* have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. On the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed. As a result, there are situations when (1) a tester writes many test cases to check something that could have been tested by only one test case, and/or (2) some parts of the back-end are not tested at all.

Therefore, black box testing has the advantage of "an unaffiliated opinion", on the one hand, and the disadvantage of "blind exploring", on the other.

Grey box testing

Grey box testing (American spelling: **gray box testing**) involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Manipulating input data and formatting output do not qualify as grey box, because the input and output are clearly outside of the "black-box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey box, as the user would not normally be able to change the data outside of the system under test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages.

Testing levels

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without impliying a specific process model. Other test levels are classified by the testing objective.

Test target

Unit testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These type of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Unit testing is also called *component testing*.

Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

System testing

System testing tests a completely integrated system to verify that it meets its requirements.

System integration testing

System integration testing verifies that a system is integrated to any external or third-party systems defined in the system requirements.

Objectives of testing

Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, to very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk.

Acceptance testing

Acceptance testing can mean one of two things:

1. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.
2. Acceptance testing is performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

Non-functional testing

Special methods exist to test non-functional aspects of software. In contrast to functional testing, which establishes the correct operation of the software (correct in that it matches the expected behavior defined in the design requirements), non-functional testing verifies that the software functions properly even when it receives invalid or unexpected inputs. Software fault injection, in the form of fuzzing, is an example of non-functional testing. Non-functional testing, especially for software, is designed to establish whether the device under test can tolerate invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-handling routines. Various commercial non-functional testing tools are linked from the software fault injection page;

there are also numerous open-source and free software tools available that perform non-functional testing.

Software performance testing and load testing

Performance testing is executed to determine how fast a system or sub-system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage. Load testing is primarily concerned with testing that can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as *endurance testing*.

Volume testing is a way to test functionality. *Stress testing* is a way to test reliability. *Load testing* is a way to test performance. There is little agreement on what the specific goals of load testing are. The terms load testing, performance testing, reliability testing, and volume testing, are often used interchangeably.

Stability testing

Stability testing checks to see if the software can continuously function well in or above an acceptable period. This activity of non-functional software testing is often referred to as load (or endurance) testing.

Usability testing

Usability testing is needed to check if the user interface is easy to use and understand.It approach towards the use of the application.

Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

Internationalization and localization

The general ability of software to be internationalized and localized can be automatically tested without actual translation, by using pseudolocalization. It will verify that the application still works, even after it has been translated into a new language or adapted for a new culture (such as different currencies or time zones).

Actual translation to human languages must be tested, too. Possible localization failures include:

- Software is often localized by translating a list of strings out of context, and the translator may choose the wrong translation for an ambiguous source string.

- Technical terminology may become inconsistent if the project is translated by several people without proper coordination or if the translator is imprudent.
- Literal word-for-word translations may sound inappropriate, artificial or too technical in the target language.
- Untranslated messages in the original language may be left hard coded in the source code.
- Some messages may be created automatically in run time and the resulting string may be ungrammatical, functionally incorrect, misleading or confusing.
- Software may use a keyboard shortcut which has no function on the source language's keyboard layout, but is used for typing characters in the layout of the target language.
- Software may lack support for the character encoding of the target language.
- Fonts and font sizes which are appropriate in the source language, may be inappropriate in the target language; for example, CJK characters may become unreadable if the font is too small.
- A string in the target language may be longer than the software can handle. This may make the string partly invisible to the user or cause the software to crash or malfunction.
- Software may lack proper support for reading or writing bi-directional text.
- Software may display images with text that wasn't localized.
- Localized operating systems may have differently-named system configuration files and environment variables and different formats for date and currency.

To avoid these and other localization problems, a tester who knows the target language must run the program with all the possible use cases for translation to see if the messages are readable, translated correctly in context and don't cause failures.

Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail, in order to test its robustness.

The testing process

Traditional CMMI or waterfall development model

A common practice of software testing is that testing is performed by an independent group of testers after the functionality is developed, before it is shipped to the customer. This practice often results in the testing phase being used as a project buffer to compensate for project delays, thereby compromising the time devoted to testing.

Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Agile or Extreme development model

In counterpoint, some emerging software disciplines such as extreme programming and the agile software development movement, adhere to a "test-driven software development" model. In this process, unit tests are written first, by the software engineers (often with pair programming in the extreme programming methodology). Of course these tests fail initially; as they are expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process). The ultimate goal of this test process is to achieve continuous deployment where software updates can be published to the public frequently.

A sample testing cycle

Although variations exist between organizations, there is a typical cycle for testing. The sample below is common among organizations employing the Waterfall development model.

- **Requirements analysis**: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.
- **Test planning**: Test strategy, test plan, testbed creation. Since many activities will be carried out during testing, a plan is needed.
- **Test development**: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.
- **Test execution**: Testers execute the software based on the plans and test documents then report any errors found to the development team.
- **Test reporting**: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
- **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be treated, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
- **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team. AKA Resolution testing.
- **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.

- **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

Automated testing

Many programming groups are relying more and more on automated testing, especially groups that use test-driven development. There are many frameworks to write tests in, and continuous integration software will run tests automatically every time code is checked into a version control system.

While automation cannot reproduce everything that a human can do (and all the ways they think of doing it), it can be very useful for regression testing. However, it does require a well-developed test suite of testing scripts in order to be truly useful.

Testing tools

Program testing and fault detection can be aided significantly by testing tools and debuggers. Testing/debug tools include features such as:

- Program monitors, permitting full or partial monitoring of program code including:
    - Instruction set simulator, permitting complete instruction level monitoring and trace facilities
    - Program animation, permitting step-by-step execution and conditional breakpoint at source level or in machine code
    - Code coverage reports
- Formatted dump or symbolic debugging, tools allowing inspection of program variables on error or at chosen points
- Automated functional GUI testing tools are used to repeat system-level tests through the GUI
- Benchmarks, allowing run-time performance comparisons to be made
- Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage

Some of these features may be incorporated into an Integrated Development Environment (IDE).

- A regression testing technique is to have a standard set of tests, which cover existing functionality that result in persistent tabular data, and to compare pre-change data to post-change data, where there should not be differences, using a tool like diffkit. Differences detected indicate unexpected functionality changes or "regression".

Measurement in software testing

Usually, quality is constrained to such topics as correctness, completeness, security, but can also include more technical requirements as described under the ISO standard ISO/IEC 9126, such as capability, reliability, efficiency, portability, maintainability, compatibility, and usability.

There are a number of frequently-used software measures, often called *metrics*, which are used to assist in determining the state of the software or the adequacy of the testing.

Testing artifacts

Software testing process can produce several artifacts.

Test plan
: A test specification is called a test plan. The developers are well aware what test plans will be executed and this information is made available to management and the developers. The idea is to make them more cautious when developing their code or making additional changes. Some companies have a higher-level document called a test strategy.

Traceability matrix
: A traceability matrix is a table that correlates requirements or design documents to test documents. It is used to change tests when the source documents are changed, or to verify that the test results are correct.

Test case
: A test case normally consists of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps (also known as actions) to follow, input, output, expected result, and actual result. Clinically defined a test case is an input and an expected result. This can be as pragmatic as 'for condition x your derived result is y', whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step, or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database, or other common repository. In a database system, you may also be able to see past test results, who generated the results, and what system configuration was used to generate those results. These past results would usually be stored in a separate table.

Test script
: The test script is the combination of a test case, test procedure, and test data. Initially the term was derived from the product of work created by automated

regression test tools. Today, test scripts can be manual, automated, or a combination of both.

Test suite
: The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Test data
: In most cases, multiple sets of values or data are used to test the same functionality of a particular feature. All the test values and changeable environmental components are collected in separate files and stored as test data. It is also useful to provide this data to the client and with the product or a project.

Test harness
: The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

Certifications

Several certification programs exist to support the professional aspirations of software testers and quality assurance specialists. No certification currently offered actually requires the applicant to demonstrate the ability to test software. No certification is based on a widely accepted body of knowledge. This has led some to declare that the testing field is not ready for certification. Certification itself cannot measure an individual's productivity, their skill, or practical knowledge, and cannot guarantee their competence, or professionalism as a tester.

Software testing certification types

- *Exam-based*: Formalized exams, which need to be passed; can also be learned by self-study [e.g., for ISTQB or QAI]
- *Education-based*: Instructor-led sessions, where each course has to be passed [e.g., International Institute for Software Testing (IIST)].

Testing certifications

- Certified Associate in Software Testing (CAST) offered by the Quality Assurance Institute (QAI)
- CATe offered by the *International Institute for Software Testing*
- Certified Manager in Software Testing (CMST) offered by the Quality Assurance Institute (QAI)
- Certified Software Tester (CSTE) offered by the Quality Assurance Institute (QAI)
- Certified Software Test Professional (CSTP) offered by the *International Institute for Software Testing*
- CSTP (TM) (Australian Version) offered by *K. J. Ross & Associates*

- ISEB offered by the Information Systems Examinations Board
- ISTQB Certified Tester, Foundation Level (CTFL) offered by the International Software Testing Qualification Board
- ISTQB Certified Tester, Advanced Level (CTAL) offered by the International Software Testing Qualification Board
- TMPF TMap Next Foundation offered by the *Examination Institute for Information Science*
- TMPA TMap Next Advanced offered by the *Examination Institute for Information Science*

Quality assurance certifications

- CMSQ offered by the *Quality Assurance Institute* (QAI).
- CSQA offered by the *Quality Assurance Institute* (QAI)
- CSQE offered by the American Society for Quality (ASQ)
- CQIA offered by the American Society for Quality (ASQ)

Controversy

Some of the major software testing controversies include:

What constitutes responsible software testing?

Members of the "context-driven" school of testing believe that there are no "best practices" of testing, but rather that testing is a set of skills that allow the tester to select or invent testing practices to suit each unique situation.

Agile vs. traditional

Should testers learn to work under conditions of uncertainty and constant change or should they aim at process "maturity"? The agile testing movement has received growing popularity since 2006 mainly in commercial circles, whereas government and military software providers use this methodology but also the traditional test-last models (e.g. in the Waterfall model).

Exploratory test vs. scripted

Should tests be designed at the same time as they are executed or should they be designed beforehand?

Manual testing vs. automated

Some writers believe that test automation is so expensive relative to its value that it should be used sparingly. More in particular, test-driven development states that developers should write unit-tests of the XUnit type before coding the functionality. The tests then can be considered as a way to capture and implement the requirements.

Software design vs. software implementation

Should testing be carried out only at the end or throughout the whole process?

Who watches the watchmen?

The idea is that any form of observation is also an interaction—the act of testing can also affect that which is being tested.

Chapter-7

Software Performance Testing

In software engineering, **performance testing** is testing that is performed, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of Performance engineering, an emerging computer science practice which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort.

Performance Testing Sub-Genres

Load Testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behavior of the application under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards any bottlenecks in the application software.

Stress Testing

Stress testing is normally used to understand the upper limits of capacity within the application landscape. This kind of test is done to determine the application's robustness in terms of extreme load and helps application administrators to determine if the application will perform sufficiently if the current load goes well above the expected maximum.

Endurance Testing (Soak Testing)

Endurance testing is usually done to determine if the application can sustain the continuous expected load. During endurance tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good or better than at the beginning of the test.

Spike Testing

Spike testing, as the name suggests is done by spiking the number of users and understanding the behavior of the application; whether performance will suffer, the application will fail, or it will be able to handle dramatic changes in load.

Configuration Testing

Configuration testing is another variation on traditional performance testing. Rather than testing for performance from the perspective of load you are testing the effects of configuration changes in the application landscape on application performance and behaviour. A common example would be experimenting with different methods of load-balancing.

Isolation Testing

Isolation testing is not unique to performance testing but a term used to describe repeating a test execution that resulted in an application problem. Often used to isolate and confirm the fault domain.

Setting performance goals

Performance testing can serve different purposes.

- It can demonstrate that the system meets performance criteria.
- It can compare two systems to find which performs better.
- Or it can measure what parts of the system or workload causes the system to perform badly.

Many performance tests are undertaken without due consideration to the setting of realistic performance goals. The first question from a business perspective should always be "why are we performance testing?". These considerations are part of the business case of the testing. Performance goals will differ depending on the application technology and purpose however they should always include some of the following:-

Concurrency / Throughput

If an application identifies end-users by some form of login procedure then a concurrency goal is highly desirable. By definition this is the largest number of concurrent application users that the application is expected to support at any given moment. The work-flow of your scripted transaction may impact true application concurrency especially if the iterative part contains the Login & Logout activity

If your application has no concept of end-users then your performance goal is likely to be based on a maximum throughput or transaction rate.

Server response time

This refers to the time taken for one application node to respond to the request of another. A simple example would be a HTTP 'GET' request from browser client to web server. In terms of response time this is what all load testing tools actually measure. It may be relevant to set server response time goals between all nodes of the application landscape.

Render response time

A difficult thing for load testing tools to deal with as they generally have no concept of what happens within a node apart from recognising a period of time where there is no activity 'on the wire'. To measure render response time it is generally necessary to include functional test scripts as part of the performance test scenario which is a feature not offered by many load testing tools.

Performance specifications

It is critical to detail performance specifications (requirements) and document them in any performance test plan. Ideally, this is done during the requirements development phase of any system development project, prior to any design effort.

However, **performance testing** is frequently not performed against a specification i.e. no one will have expressed what the maximum acceptable response time for a given population of users should be. Performance testing is frequently used as part of the process of performance profile tuning. The idea is to identify the "weakest link" – there is inevitably a part of the system which, if it is made to respond faster, will result in the overall system running faster. It is sometimes a difficult task to identify which part of the system represents this critical path, and some test tools include (or can have add-ons that provide) instrumentation that runs on the server (agents) and report transaction times, database access times, network overhead, and other server monitors, which can be analyzed together with the raw performance statistics. Without such instrumentation one might have to have someone crouched over Windows Task Manager at the server to see how much CPU load the performance tests are generating (assuming a Windows system is under test).

There is an apocryphal story of a company that spent a large amount optimizing their software without having performed a proper analysis of the problem. They ended up rewriting the system's 'idle loop', where they had found the system spent most of its time, but even having the most efficient idle loop in the world obviously didn't improve overall performance one iota!

Performance testing can be performed across the web, and even done in different parts of the country, since it is known that the response times of the internet itself vary regionally. It can also be done in-house, although routers would then need to be configured to introduce the lag what would typically occur on public networks. Loads should be introduced to the system from realistic points. For example, if 50% of a system's user base will be accessing the system via a 56K modem connection and the other half over a T1, then the load injectors (computers that simulate real users) should either inject load over the same connections (ideal) or simulate the network latency of such connections, following the same user profile.

It is always helpful to have a statement of the likely peak numbers of users that might be expected to use the system at peak times. If there can also be a statement of what constitutes the maximum allowable 95 percentile response time, then an injector configuration could be used to test whether the proposed system met that specification.

Questions to ask

Performance specifications should ask the following questions, at a minimum:

- In detail, what is the performance test scope? What subsystems, interfaces, components, etc. are in and out of scope for this test?
- For the user interfaces (UI's) involved, how many concurrent users are expected for each (specify peak vs. nominal)?
- What does the target system (hardware) look like (specify all server and network appliance configurations)?
- What is the Application Workload Mix of each application component? (for example: 20% login, 40% search, 30% item select, 10% checkout).
- What is the System Workload Mix? [Multiple workloads may be simulated in a single performance test] (for example: 30% Workload A, 20% Workload B, 50% Workload C)
- What are the time requirements for any/all backend batch processes (specify peak vs. nominal)?

Pre-requisites for Performance Testing

A stable build of the application which must resemble the Production environment as close to possible.

The performance testing environment should not be clubbed with User acceptance testing (UAT) or development environment. This is dangerous as if an UAT or Integration test or other tests are going on in the same environment, then the results obtained from the performance testing may not be reliable. As a best practice it is always advisable to have a separate performance testing environment resembling the production environment as much as possible.

Test conditions

In performance testing, it is often crucial (and often difficult to arrange) for the test conditions to be similar to the expected actual use. This is, however, not entirely possible in actual practice. The reason is that the workloads of production systems have a random nature, and while the test workloads do their best to mimic what may happen in the production environment, it is impossible to exactly replicate this workload variability - except in the most simple system.

Loosely-coupled architectural implementations (e.g.: SOA) have created additional complexities with performance testing. Enterprise services or assets (that share a common infrastructure or platform) require coordinated performance testing (with all consumers creating production-like transaction volumes and load on shared infrastructures or platforms) to truly replicate production-like states. Due to the complexity and financial and time requirements around this activity, some organizations now employ tools that can monitor and create production-like conditions (also referred as "noise") in their performance testing environments (PTE) to understand capacity and resource requirements and verify / validate quality attributes.

Timing

It is critical to the cost performance of a new system, that performance test efforts begin at the inception of the development project and extend through to deployment. The later a performance defect is detected, the higher the cost of remediation. This is true in the case of functional testing, but even more so with performance testing, due to the end-to-end nature of its scope.

Tools

In the diagnostic case, software engineers use tools such as profilers to measure what parts of a device or software contributes most to the poor performance or to establish throughput levels (and thresholds) for maintained acceptable response time.

Myths of Performance Testing

Some of the very common myths are given below.
1. Performance Testing is done to break the system.

Stress Testing is done to understand the break point of the system. Otherwise normal load testing is generally done to understand the behavior of the application under the expected user load. Depending on other requirements, such as expectation of spike load, continued load for an extended period of time would demand spike, endurance soak or stress testing.

2. Performance Testing should only be done after the System Integration Testing

Although this is mostly the norm in the industry, performance testing can also be done while the initial development of the application is taking place. This kind of approach is known as the **Early Performance Testing**. This approach would ensure a holistic development of the application keeping the performance parameters in mind. Thus the finding of a performance bug just before the release of the application and the cost involved in rectifying the bug is reduced to a great extent.

3. Performance Testing only involves creation of scripts and any application changes would cause a simple refactoring of the scripts.

Performance Testing in itself is an evolving science in the Software Industry. Scripting itself although important, is only one of the components of the performance testing. The major challenge for any performance tester is to determine the type of tests needed to execute and analyzing the various performance counters to determine the performance bottleneck.

The other segment of the myth concerning the change in application would result only in little refactoring in the scripts is also untrue as any form of change on the UI especially in the Web protocol would entail complete re-development of the scripts from scratch. This problem becomes bigger if the protocols involved include Web Services, Siebel, Citrix, and SAP.

Technology

Performance testing technology employs one or more PCs or Unix servers to act as injectors – each emulating the presence of numbers of users and each running an automated sequence of interactions (recorded as a script, or as a series of scripts to emulate different types of user interaction) with the host whose performance is being tested. Usually, a separate PC acts as a test conductor, coordinating and gathering metrics from each of the injectors and collating performance data for reporting purposes. The usual sequence is to ramp up the load – starting with a small number of virtual users and increasing the number over a period to some maximum. The test result shows how the performance varies with the load, given as number of users vs response time. Various tools, are available to perform such tests. Tools in this category usually execute a suite of tests which will emulate real users against the system. Sometimes the results can reveal oddities, e.g., that while the average response time might be acceptable, there are outliers of a few key transactions that take considerably longer to complete – something that might be caused by inefficient database queries, pictures etc.

Performance testing can be combined with stress testing, in order to see what happens when an acceptable load is exceeded –does the system crash? How long does it take to recover if a large load is reduced? Does it fail in a way that causes collateral damage?

Analytical Performance Modeling is a method to model the behaviour of an application in a spreadsheet. The model is fed with measurements of transaction resource demands (CPU, disk I/O, LAN, WAN), weighted by the transaction-mix (business transactions per hour). The weighted transaction resource demands are added-up to obtain the hourly resource demands and divided by the hourly resource capacity to obtain the resource loads. Using the responsetime formula ($R=S/(1-U)$, R=responsetime, S=servicetime, U=load), responsetimes can be calculated and calibrated with the results of the performance tests. Analytical performance modelling allows evaluation of design options and system sizing based on actual or anticipated business usage. It is therefore much faster and cheaper than performance testing, though it requires thorough understanding of the hardware platforms.

Tasks to undertake

Tasks to perform such a test would include:

- Decide whether to use internal or external resources to perform the tests, depending on inhouse expertise (or lack thereof)
- Gather or elicit performance requirements (specifications) from users and/or business analysts
- Develop a high-level plan (or project charter), including requirements, resources, timelines and milestones
- Develop a detailed performance test plan (including detailed scenarios and test cases, workloads, environment info, etc.)
- Choose test tool(s)
- Specify test data needed and charter effort (often overlooked, but often the death of a valid performance test)
- Develop proof-of-concept scripts for each application/component under test, using chosen test tools and strategies
- Develop detailed performance test project plan, including all dependencies and associated timelines
- Install and configure injectors/controller
- Configure the test environment (ideally identical hardware to the production platform), router configuration, quiet network (we don’t want results upset by other users), deployment of server instrumentation, database test sets developed, etc.
- Execute tests – probably repeatedly (iteratively) in order to see whether any unaccounted for factor might affect the results
- Analyze the results - either pass/fail, or investigation of critical path and recommendation of corrective action

Methodology

Performance Testing Web Applications Methodology

According to the Microsoft Developer Network the Performance Testing Methodology consists of the following activities:

- **Activity 1. Identify the Test Environment.** Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software, and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. In some situations, this process must be revisited periodically throughout the project's life cycle.
- **Activity 2. Identify Performance Acceptance Criteria.** Identify the response time, throughput, and resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern, and resource utilization is a system concern. Additionally, identify project success criteria that may not be captured by those goals and constraints; for example, using performance tests to evaluate what combination of configuration settings will result in the most desirable performance characteristics.
- **Activity 3. Plan and Design Tests.** Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected. Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed.
- **Activity 4. Configure the Test Environment.** Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test. Ensure that the test environment is instrumented for resource monitoring as necessary.
- **Activity 5. Implement the Test Design.** Develop the performance tests in accordance with the test design.
- **Activity 6. Execute the Test.** Run and monitor your tests. Validate the tests, test data, and results collection. Execute validated tests for analysis while monitoring the test and the test environment.
- **Activity 7. Analyze Results, Tune, and Retest.** Analyse, Consolidate and share results data. Make a tuning change and retest. Improvement or degradation? Each improvement made will return smaller improvement than the previous improvement. When do you stop? When you reach a CPU bottleneck, the choices then are either improve the code or add more CPU.

Chapter-8

Graphical user Interface Testing

In software engineering, **graphical user interface testing** is the process of testing a product's graphical user interface to ensure it meets its written specifications. This is normally done through the use of a variety of test cases.

Test Case Generation

To generate a 'good' set of test cases, the test designers must be certain that their suite covers all the functionality of the system and also has to be sure that the suite fully exercises the GUI itself. The difficulty in accomplishing this task is twofold: one has to deal with domain size and then one has to deal with sequences. In addition, the tester faces more difficulty when they have to do regression testing.

The size problem can be easily illustrated. Unlike a CLI (command line interface) system, a GUI has many operations that need to be tested. A relatively small program such as Microsoft WordPad has 325 possible GUI operations. In a large program, the number of operations can easily be an order of magnitude larger.

The second problem is the sequencing problem. Some functionality of the system may only be accomplishable by following some complex sequence of GUI events. For example, to open a file a user may have to click on the File Menu and then select the Open operation, and then use a dialog box to specify the file name, and then focus the application on the newly opened window. Obviously, increasing the number of possible operations increases the sequencing problem exponentially. This can become a serious issue when the tester is creating test cases manually.

Regression testing becomes a problem with GUIs as well. This is because the GUI may change significantly across versions of the application, even though the underlying application may not. A test designed to follow a certain path through the GUI may not be able to follow that path since a button, menu item, or dialog may have changed location or appearance.

These issues have driven the GUI testing problem domain towards automation. Many different techniques have been proposed to automatically generate test suites that are complete and that simulate user behavior.

Most of the techniques used to test GUIs attempt to build on techniques previously used to test CLI programs. However, most of these have scaling problems when they are applied to GUI's. For example, Finite State Machine-based modeling — where a system is modeled as a finite state machine and a program is used to generate test cases that exercise all states — can work well on a system that has a limited number of states but may become overly complex and unwieldy for a GUI.

Planning and artificial intelligence

A novel approach to test suite generation, adapted from a **CLI** technique involves using a planning system. Planning is a well-studied technique from the artificial intelligence (AI) domain that attempts to solve problems that involve four parameters:

- an initial state,
- a goal state,
- a set of operators, and
- a set of objects to operate on.

Planning systems determine a path from the initial state to the goal state by using the operators. An extremely simple planning problem would be one where you had two words and one operation called 'change a letter' that allowed you to change one letter in a word to another letter – the goal of the problem would be to change one word into another.

For GUI testing, the problem is a bit more complex. In the authors used a planner called IPP to demonstrate this technique. The method used is very simple to understand. First, the systems UI is analyzed to determine what operations are possible. These operations become the operators used in the planning problem. Next an initial system state is determined. Next a goal state is determined that the tester feels would allow exercising of the system. Lastly the planning system is used to determine a path from the initial state to the goal state. This path becomes the test plan.

Using a planner to generate the test cases has some specific advantages over manual generation. A planning system, by its very nature, generates solutions to planning problems in a way that is very beneficial to the tester:

1. The plans are always valid. What this means is that the output of the system can be one of two things, a valid and correct plan that uses the operators to attain the goal state or no plan at all. This is beneficial because much time can be wasted when manually creating a test suite due to invalid test cases that the tester thought would work but didn't.

2. A planning system pays attention to order. Often to test a certain function, the test case must be complex and follow a path through the GUI where the operations are performed in a specific order. When done manually, this can lead to errors and also can be quite difficult and time consuming to do.
3. Finally, and most importantly, a planning system is goal oriented. What this means and what makes this fact so important is that the tester is focusing test suite generation on what is most important, testing the functionality of the system.

When manually creating a test suite, the tester is more focused on how to test a function (i. e. the specific path through the GUI). By using a planning system, the path is taken care of and the tester can focus on what function to test. An additional benefit of this is that a planning system is not restricted in any way when generating the path and may often find a path that was never anticipated by the tester. This problem is a very important one to combat.

Another interesting method of generating GUI test cases uses the theory that good GUI test coverage can be attained by simulating a novice user. One can speculate that an expert user of a system will follow a very direct and predictable path through a GUI and a novice user would follow a more random path. The theory therefore is that if we used an expert to test the GUI, many possible system states would never be achieved. A novice user, however, would follow a much more varied, meandering and unexpected path to achieve the same goal so it's therefore more desirable to create test suites that simulate novice usage because they will test more.

The difficulty lies in generating test suites that simulate 'novice' system usage. Using Genetic algorithms is one proposed way to solve this problem. Novice paths through the system are not random paths. First, a novice user will learn over time and generally won't make the same mistakes repeatedly, and, secondly, a novice user is not analogous to a group of monkeys trying to type Hamlet, but someone who is following a plan and probably has some domain or system knowledge.

Genetic algorithms work as follows: a set of 'genes' are created randomly and then are subjected to some task. The genes that complete the task best are kept and the ones that don't are discarded. The process is again repeated with the surviving genes being replicated and the rest of the set filled in with more random genes. Eventually one gene (or a small set of genes if there is some threshold set) will be the only gene in the set and is naturally the best fit for the given problem.

For the purposes of the GUI testing, the method works as follows. Each gene is essentially a list of random integer values of some fixed length. Each of these genes represents a path through the GUI. For example, for a given tree of widgets, the first value in the gene (each value is called an allele) would select the widget to operate on, the following alleles would then fill in input to the widget depending on the number of possible inputs to the widget (for example a pull down list box would have one input…the selected list value). The success of the genes are scored by a criterion that rewards the best 'novice' behavior.

The system to do this testing described in can be extended to any windowing system but is described on the X window system. The X Window system provides functionality (via XServer and the editors' protocol) to dynamically send GUI input to and get GUI output from the program without directly using the GUI. For example, one can call XSendEvent() to simulate a click on a pull-down menu, and so forth. This system allows researchers to automate the gene creation and testing so for any given application under test, a set of novice user test cases can be created.

Running the test cases

At first the strategies were migrated and adapted from the CLI testing strategies. A popular method used in the CLI environment is capture/playback. Capture playback is a system where the system screen is “captured” as a bitmapped graphic at various times during system testing. This capturing allowed the tester to “play back” the testing process and compare the screens at the output phase of the test with expected screens. This validation could be automated since the screens would be identical if the case passed and different if the case failed.

Using capture/playback worked quite well in the CLI world but there are significant problems when one tries to implement it on a GUI-based system. The most obvious problem one finds is that the screen in a GUI system may look different while the state of the underlying system is the same, making automated validation extremely difficult. This is because a GUI allows graphical objects to vary in appearance and placement on the screen. Fonts may be different, window colors or sizes may vary but the system output is basically the same. This would be obvious to a user, but not obvious to an automated validation system.

To combat this and other problems, testers have gone ‘under the hood’ and collected GUI interaction data from the underlying windowing system. By capturing the window ‘events’ into logs the interactions with the system are now in a format that is decoupled from the appearance of the GUI. Now, only the event streams are captured. There is some filtering of the event streams necessary since the streams of events are usually very detailed and most events aren’t directly relevant to the problem. This approach can be made easier by using an MVC architecture for example and making the view (i. e. the GUI here) as simple as possible while the model and the controller hold all the logic. Another approach is to use the software's built-in assistive technology, to use an HTML interface or a three-tier architecture that makes it also possible to better separate the user interface from the rest of the application.

Another way to run tests on a GUI is to build a driver into the GUI so that commands or events can be sent to the software from another program. This method of directly sending events to and receiving events from a system is highly desirable when testing, since the input and output testing can be fully automated and user error is eliminated.

Chapter-9

Evidence-based Medicine

Evidence-based medicine (EBM) or **evidence-based practice** (EBP) aims to apply the best available evidence gained from the scientific method to clinical decision making. It seeks to assess the strength of evidence of the risks and benefits of treatments (including lack of treatment) and diagnostic tests. Evidence quality can range from meta-analyses and systematic reviews of double-blind, placebo-controlled clinical trials at the top end, down to conventional wisdom at the bottom.

EBM/EBP recognizes that many aspects of health care depend on individual factors such as quality- and value-of-life judgments, which are only partially subject to scientific methods. EBP, however, seeks to clarify those parts of medical practice that are in principle subject to scientific methods and to apply these methods to ensure the best *prediction* of outcomes in medical treatment, even as debate continues about which outcomes are desirable.

Because this approach is used in allied related fields, including dentistry, nursing, and psychology, *evidenced-based practice* is a more encompassing term.

Classification

Two types of evidence-based practice have been proposed.

Evidence-based guidelines

Evidence-based guidelines (EBG) is the practice of evidence-based medicine at the organizational or institutional level. This includes the production of guidelines, policy, and regulations. This approach has also been called evidence based healthcare.

Evidence-based individual decision making

Evidence-based individual decision (EBID) making is evidence-based medicine as practiced by the individual health care provider. There is concern that current evidence-based medicine focuses excessively on EBID. The American Academy of Family

Physicians (AAFP) says that DynaMed may be of assistance to family physicians in answering clinical questions with high-quality evidence.

Process and progress

Using techniques from science, engineering, and statistics, such as the systematic review of medical literature, meta-analysis, risk-benefit analysis, and randomized controlled trials (RCTs), EBM aims for the ideal that healthcare professionals should make "conscientious, explicit, and judicious use of current best evidence" in their everyday practice. *Ex cathedra* statements by the "medical expert" are considered to be least valid form of evidence. All "experts" are now expected to reference their pronouncements to scientific studies.

The systematic review of published research studies is a major method used for evaluating particular treatments. The Cochrane Collaboration is one of the best-known, respected examples of systematic reviews. Like other collections of systematic reviews, it requires authors to provide a detailed and repeatable plan of their literature search and evaluations of the evidence. Once all the best evidence is assessed, treatment is categoried as "likely to be beneficial", "likely to be harmful", or "evidence did not support either benefit or harm".

A 2007 analysis of 1016 systematic reviews from all 50 Cochrane Collaboration Review Groups found that 44% of the reviews concluded that the intervention was "likely to be beneficial", 7% concluded that the intervention was "likely to be harmful", and 49% concluded that evidence "did not support either benefit or harm". 96% recommended further research. A 2001 review of 160 Cochrane systematic reviews (excluding complementary treatments) in the 1998 database revealed that, according to two readers, 41.3% concluded positive or possibly positive effect, 20% concluded evidence of no effect, 8.1% concluded net harmful effects, and 21.3% of the reviews concluded insufficient evidence. A review of 145 alternative medicine Cochrane reviews using the 2004 database revealed that 38.4% concluded positive effect or possibly positive (12.4%) effect, 4.8% concluded no effect, 0.69% concluded harmful effect, and 56.6% concluded insufficient evidence.

Generally, there are three distinct, but interdependent, areas of EBM. The first is to treat individual patients with acute or chronic pathologies by treatments supported in the most scientifically valid medical literature. Thus, medical practitioners would select treatment options for specific cases based on the best research for each patient they treat. The second area is the systematic review of medical literature to evaluate the best studies on specific topics. This process can be very human-centered, as in a journal club, or highly technical, using computer programs and information techniques such as data mining. Increased use of information technology turns large volumes of information into practical guides. Finally, evidence-based medicine can be understood as a medical "movement" in which advocates work to popularize the method and usefulness of the practice in the public, patient communities, educational institutions, and continuing education of practicing professionals.

Ranking the quality of evidence

Evidence-based medicine categorizes different types of clinical evidence and rates or grades them according to the strength of their freedom from the various biases that beset medical research. For example, the strongest evidence for therapeutic interventions is provided by systematic review of randomized, triple-blind, placebo-controlled trials with allocation concealment and complete follow-up involving a homogeneous patient population and medical condition. In contrast, patient testimonials, case reports, and even expert opinion have little value as proof because of the placebo effect, the biases inherent in observation and reporting of cases, difficulties in ascertaining who is an expert, and more.

US Preventive Services Task Force

Systems to stratify evidence by quality have been developed, such as this one by the U.S. Preventive Services Task Force for ranking evidence about the effectiveness of treatments or screening:

- Level I: Evidence obtained from at least one properly designed randomized controlled trial.
- Level II-1: Evidence obtained from well-designed controlled trials without randomization.
- Level II-2: Evidence obtained from well-designed cohort or case-control analytic studies, preferably from more than one center or research group.
- Level II-3: Evidence obtained from multiple time series with or without the intervention. Dramatic results in uncontrolled trials might also be regarded as this type of evidence.
- Level III: Opinions of respected authorities, based on clinical experience, descriptive studies, or reports of expert committees.

National Health Service

The UK National Health Service uses a similar system with categories labeled A, B, C, and D. The above Levels are only appropriate for treatment or interventions; different types of research are required for assessing diagnostic accuracy or natural history and prognosis, and hence different "levels" are required. For example, the Oxford Centre for Evidence-based Medicine suggests levels of evidence (LOE) according to the study designs and critical appraisal of prevention, diagnosis, prognosis, therapy, and harm studies:

- Level A: Consistent Randomised Controlled Clinical Trial, cohort study, all or none, clinical decision rule validated in different populations.
- Level B: Consistent Retrospective Cohort, Exploratory Cohort, Ecological Study, Outcomes Research, case-control study; or extrapolations from level A studies.
- Level C: Case-series study or extrapolations from level B studies.

- Level D: Expert opinion without explicit critical appraisal, or based on physiology, bench research or first principles.

GRADE Working Group

A newer system is by the GRADE Working Group and takes in account more dimensions than just the quality of medical evidence. "Extrapolations" are where data is used in a situation which has potentially clinically important differences than the original study situation. Thus, the quality of evidence to support a clinical decision is a combination of the quality of research data and the clinical 'directness' of the data.

Despite the differences between systems, the purposes are the same: to guide users of clinical research information about which studies are likely to be most valid. However, the individual studies still require careful critical appraisal.

Note: The all or none principle is met when all patients died before the Rx became available, but some now survive on it; or when some patients died before the Rx became available, but none now die on it.

Categories of recommendations

In guidelines and other publications, recommendation for a clinical service is classified by the balance of risk versus benefit of the service *and* the level of evidence on which this information is based. The U.S. Preventive Services Task Force uses:

- Level A: Good scientific evidence suggests that the benefits of the clinical service substantially outweigh the potential risks. Clinicians should discuss the service with eligible patients.
- Level B: At least fair scientific evidence suggests that the benefits of the clinical service outweighs the potential risks. Clinicians should discuss the service with eligible patients.
- Level C: At least fair scientific evidence suggests that there are benefits provided by the clinical service, but the balance between benefits and risks are too close for making general recommendations. Clinicians need not offer it unless there are individual considerations.
- Level D: At least fair scientific evidence suggests that the risks of the clinical service outweighs potential benefits. Clinicians should not routinely offer the service to asymptomatic patients.
- Level I: Scientific evidence is lacking, of poor quality, or conflicting, such that the risk versus benefit balance cannot be assessed. Clinicians should help patients understand the uncertainty surrounding the clinical service.

Statistical measures

Evidence-based medicine attempts to express clinical benefits of tests and treatments using mathematical methods. Tools used by practitioners of evidence-based medicine include:

Likelihood ratio

The pretest odds of a particular diagnosis, multiplied by the likelihood ratio, determines the post-test odds. (Odds can be calculated from, and then converted to, the [more familiar] probability.) This reflects Bayes' theorem. The differences in likelihood ratio between clinical tests can be used to prioritize clinical tests according to their usefulness in a given clinical situation.

AUC-ROC

The area under the receiver operating characteristic curve (AUC-ROC) reflects the relationship between sensitivity and specificity for a given test. High-quality tests will have an AUC-ROC approaching 1, and high-quality publications about clinical tests will provide information about the AUC-ROC. Cutoff values for positive and negative tests can influence specificity and sensitivity, but they do not affect AUC-ROC.

Number needed to treat / harm

Number needed to treat or Number needed to harm are ways of expressing the effectiveness and safety of an intervention in a way that is clinically meaningful. In general, NNT is always computed with respect to two treatments A and B, with A typically a drug and B a placebo (in our example above, A is a 5-year treatment with the hypothetical drug, and B is no treatment). A defined endpoint has to be specified (in our example: the appearance of colon cancer in the 5 year period). If the probabilities pA and pB of this endpoint under treatments A and B, respectively, are known, then the NNT is computed as 1/(pB-pA). The NNT for breast mammography is 285; that is, 285 mammograms need to be performed to diagnose one breast cancer. As another example, an NNT of 4 means if 4 patients are treated, only one would respond.

An NNT of 1 is the most effective and means each patient treated responds, e.g., in comparing antibiotics with placebo in the eradication of *Helicobacter pylori*. An NNT of 2 or 3 indicates that a treatment is quite effective (with one patient in 2 or 3 responding to the treatment). An NNT of 20 to 40 can still be considered clinically effective.

Quality of clinical trials

Evidence-based medicine attempts to objectively evaluate the quality of clinical research by critically assessing techniques reported by researchers in their publications.

- Trial design considerations. High-quality studies have clearly defined eligibility criteria, and have minimal missing data.
- Generalizability considerations. Studies may only be applicable to narrowly defined patient populations, and may not be generalizable to clinical practice.
- Followup. Sufficient time for defined outcomes to occur can influence the study outcomes and the statistical power of a study to detect differences between a treatment and control arm.
- Power. A mathematical calculation can determine if the number of patients is sufficient to detect a difference between treatment arms. A negative study may reflect a lack of benefit, or simply a lack of sufficient quantities of patients to detect a difference.

Limitations

Although evidence-based medicine is becoming regarded as the "gold standard" for clinical practice there are a number of limitations and criticisms of its use.

Ethics

In some cases, such as in open-heart surgery, conducting randomized, placebo-controlled trials is commonly considered to be unethical, although observational studies may address these problems to some degree.

Cost

The types of trials considered "gold standard" (i.e. large randomized double-blind placebo-controlled trials) are expensive, so that funding sources play a role in what gets investigated. For example, public authorities may tend to fund preventive medicine studies to improve public health, while pharmaceutical companies fund studies intended to demonstrate the efficacy and safety of particular drugs.

Generalizability

Furthermore, evidence-based guidelines do not remove the problem of extrapolation to different populations or longer timeframes. Even if several top-quality studies are available, questions always remain about how far, and to which populations, their results are "generalizable". Furthermore, skepticism about results may always be extended to areas not explicitly covered: for example, a drug may influence a "secondary endpoint" such as test result (blood pressure, glucose, or cholesterol levels) without having the power to show that it decreases overall mortality or morbidity in a population.

The quality of studies performed varies, making it difficult to compare them and generalize about the results.

Certain groups have been historically under-researched (racial minorities and people with many co-morbid diseases), and thus the literature is sparse in areas that do not allow for generalizing.

Publication bias

It is recognised that not all evidence is made accessible, that this can limit the effectiveness of any approach, and that efforts to reduce publication bias and retrieval bias is required.

Failure to publish negative trials is the most obvious gap, and moves to register all trials at the outset, and then to pursue their results, are underway. Changes in publication methods, particularly related to the Web, should reduce the difficulty of obtaining publication for a paper on a trial that concludes it did not prove anything new, including its starting hypothesis.

Treatment effectiveness reported from clinical studies may be higher than that achieved in later routine clinical practice due to the closer patient monitoring during trials that leads to much higher compliance rates.

The studies that are published in medical journals may not be representative of all the studies that are completed on a given topic (published and unpublished) or may be unreliable due to conflicts of interest. Thus the array of evidence available on particular therapies may not be well-represented in the literature. A 2004 statement by the International Committee of Medical Journal Editors (that they will refuse to publish clinical trial results if the trial was not recorded publicly at its outset) may help with this, although this has not yet been implemented.

Ghost writers

Populations, clinical experience, and dubious diagnoses

EBM applies to groups of people but this does not preclude clinicians from using their personal experience in deciding how to treat the person in front of them. In *The limits of evidence-based medicine*, Tonelli advises that "the knowledge gained from clinical research does not directly answer the primary clinical question of what is best for the patient at hand." and suggests that evidence-based medicine should not discount the value of clinical experience.

David Sackett writes that "the practice of evidence based medicine means integrating individual clinical expertise with the best available external clinical evidence from systematic research".

Political criticism

There is a good deal of criticism of evidence based medicine, which is suspected of being — as against what the phrase suggests — in essence a tool not so much for medical science as for health managers, who want to introduce managerialist techniques into medical administration. Thus Dr Michael Fitzpatrick writes: "To some of its critics, in its disparagement of theory and its crude number-crunching, EBM marks a return to 'empiricist quackery' in medical practice . Its main appeal, as Singh and Ernst suggest, is to health economists, policymakers and managers, to whom it appears useful for measuring performance and rationing resources."

In psychiatry

Standard knowledge about mental illnesses, such as the Diagnostic and Statistical Manual of Mental Disorders, have been criticized as incompletely justified by evidence. In many cases, it is unknown whether a particular "disease" has one, several, or no underlying biological causes (controversy arising over whether some diseases are merely an artifact of the attempt to construct a unified classification scheme, rather than a "real" disease).

While some experts point to statistics in support of the idea that a lack of adoption of research findings results in suboptimal treatment for many patients, others emphasize the importance of the skill of the practitioner and the customization of the treatment to fit individual needs. There is some controversy over whether mental illnesses is too complex for broad population studies to be helpful.

History

Traces of evidence-based medicine's origin can be found in ancient Greece, Although testing medical interventions for efficacy has existed since the time of Avicenna's *The Canon of Medicine* in the 11th century, it was only in the 20th century that this effort evolved to impact almost all fields of health care and policy. Professor Archie Cochrane, a Scottish epidemiologist, through his book *Effectiveness and Efficiency: Random Reflections on Health Services* (1972) and subsequent advocacy, caused increasing acceptance of the concepts behind evidence-based practice. Cochrane's work was honoured through the naming of centres of evidence-based medical research — *Cochrane Centres* — and an international organization, the Cochrane Collaboration. The explicit methodologies used to determine "best evidence" were largely established by the McMaster University research group led by David Sackett and Gordon Guyatt. Guyatt later coined the term “evidence-based” in 1990. The term "evidence-based medicine" first appeared in the medical literature in 1992 in a paper by Guyatt *et al.* Relevant journals include the British Medical Journal's *Clinical Evidence*, the *Journal Of Evidence-Based Healthcare* and *Evidence Based Health Policy*. All of these were co-founded by Anna Donald, an Australian pioneer in the discipline.

EBM and ethics of experimental or risky treatments

Insurance companies in the United States and public insurers in other countries usually wait for drug use approval based on evidence-based guidelines before funding a treatment. Where approval for a drug has been given, and subsequent evidence based findings indicating that a drug may be less safe than originally anticipated, some insurers in the U.S. have reacted very cautiously and withdrawn funding. For example, an older generic statin drug had been shown to reduce mortality, but a newer and much more expensive statin drug was found to lower cholesterol more effectively. However, evidence came to light about safety concerns with the new drug which caused some insurers to stop funding it even though marketing approval was not withdrawn. Some people are willing to take their chances to gamble their health on the success of new drugs or old drugs in new situations which may not yet have been fully tested in clinical trials. However insurance companies are reluctant to take on the job of funding such treatments, preferring instead to take the safer route of awaiting the results of clinical testing and leaving the funding of such trials to the manufacturer seeking a license.

Sometimes caution errs in the other direction. Kaiser Permanente did not change its methods of evaluating whether or not new therapies were too "experimental" to be covered until it was successfully sued twice: once for delaying in vitro fertilization treatments for two years after the courts determined that scientific evidence of efficacy and safety had reached the "reasonable" stage; and in another case where Kaiser refused to pay for liver transplantation in infants when it had already been shown to be effective in adults, on the basis that use in infants was still "experimental." Here again, the problem of induction plays a key role in arguments.

Application of the evidence based model on other public policy matters

There has been discussion of applying what has been learned from EBM to public policy. In his 1996 inaugural speech as President of the Royal Statistical Society, Adrian Smith held out evidence-based medicine as an exemplar for *all* public policy. He proposed that "evidence-based policy" should be established for education, prisons and policing policy and all areas of government.

Chapter-10

International Healthcare Accreditation

Due to the near-universal desire for quality healthcare, there is a growing interest in **international healthcare accreditation**. Providing healthcare, especially of an adequate standard, is a complex and challenging process. Healthcare is a vital and emotive issue—its importance pervades all aspects of societies, and it has medical, dental, social, political, ethical, business, and financial ramifications. In any part of the world healthcare services can be provided either by the public sector or by the private sector, or by a combination of both, and the site of delivery of healthcare can be located in hospitals or be accessed through practitioners working in the community, such as general medical practitioners and dentists.

This is occurring in most parts of the developed world in a setting in which people are expressing ever-greater expectations of hospitals and healthcare services. This trend is especially strong where socialised medical systems exist. For example, in the European Union "... patients have ever-greater expectations of what health systems ought to deliver," although there has been a "... continuous rise in costs of services determined by scientific and technological innovation." And in one particular EU member state, the United Kingdom, "... People are going to increase demand and they have also got an increased expectation of what the NHS can deliver." Interestingly, the USA manifests some differences here, and is an unusual and distinct oddity among developed Western countries. In 2007, 45.7 million of the overall US population (i.e. 15.3%) had no health insurance whatsoever yet in 2007 the USA spent nearly $2.3 trillion on healthcare, or 16% of the country's gross domestic product, more than twice as much per capita as the OECD average. Because of this, some US citizens are having to look outside of their country to find affordable healthcare, through the medium of medical tourism, also known as "Global Healthcare".

Apart from using hospitals and healthcare services to regain their health if it has become impaired, or to prevent ill health occurring in the first place, people the world over may also use them for a wide variety of other services, for example "improving upon nature" (e.g. cosmetic surgery, gender re-assignment surgery or acquiring help to overcome difficulties with becoming a parent (e.g. infertility treatment).

Healthcare and hospital accreditation

Fundamentally healthcare and hospital accreditation is about improving how care is delivered to patients and the quality of the care they receive. Accreditation has been defined as *"A self-assessment and external peer assessment process used by health care organisations to accurately assess their level of performance in relation to established standards and to implement ways to continuously improve"* Interest in hospital accreditation ascends as far as the World Health Organisation. Accreditation is one important component in patient safety.

In the USA in the early 20th century, there was concern over how to best create an appropriate environment in which clinicians could work. Standards to improve the control of the hospital environment were thus generated, and these subsequently grew into accreditation schemes with the remit to facilitate and improve organisational development. Part of the process is not only about assessing quality, but also about promoting and improving quality. Similar accreditation schemes were soon developed elsewhere in the world.

In countries such as the United Kingdom, the USA, Australia, New Zealand and Canada, sophisticated accreditation groups have grown up to survey hospitals (and, in some cases, healthcare in the community). Furthermore, other accreditation groups have been set up with openly declared remits to look after just one particular area of healthcare, such as laboratory medicine or psychiatric services or sexual health.

Accreditation systems are structured so as to provide objective measures for the external evaluation of quality and quality management. Accreditation schemes should ideally focus primarily on the patient and their pathway through the healthcare system – this includes how they access care, how they are cared for after discharge from hospital, and the quality of the services provided for them. At the heart of these schemes is a list of standards which, ideally, serve to assess evaluate in a systematic and comprehensive way the standards of professional performance in a hospital. This includes not only hand-on patient care but also training and education of staff, credentials, clinical governance and audit, research activity, ethical standards etc. The standards can also be used internally by hospitals to develop and improve their quality standards and quality management. Some international accreditation schemes believe that the standards applied should be fixed and are non-negotiable, while others operate a system of negotiation over standards - however, whatever approach is taken the every aspect of the process should be evidence-based.

International standardization groups also exist, but it must be pointed out that the mere achieving of set standards is not the only factor involved in quality accreditation - there is also the significant matter of the incorporating into participating hospitals systems of self-examination, problem solving and self-improvement, and hence there is more to accreditation than following some sort of overall "standardization" process.

As governments and the general public have increasingly come to demand more and more openness about health care and its delivery, including and especially hospital quality and safety and the clinical performance of doctors, and these accreditation systems have generally adapted to fulfill this extended role.

However, accreditation should ideally be independent of governmental control, and accreditation groups should assess hospitals “holistically”, and not just some isolated facet of the hospital’s activities or services such as the laboratories, pharmacy services, infection control, financial health or information technology services (indeed, partial accreditation of this type should be publicly acknowledged as such by both the accreditation scheme and the hospital). The best accreditation schemes also assess academic and intellectual activity (such as teaching and research) within those hospitals that they survey and have a clear and declared interest in medical ethics.

In some parts of the world, accessing healthcare can be very expensive, even prohibitively so. While some countries have elected to provide comprehensive healthcare services for all of their populations, others appear to be satisfied with leaving portions of their population without access to healthcare. When it comes to who pays the bills for healthcare, it may be the government or it may be the individual (sometimes either by direct payment, and sometimes through employer-run schemes, insurance companies etc.), or a combination of both. However, healthcare can never be truly “free” – someone somewhere will always have to pay, and the payer will always want the best value for money possible. "Affordability" of healthcare can be the insurmountable hurdle for some human beings. Value for money is hence another factor in assessing the true quality of healthcare.

Background

A number of larger countries engage in hospital accreditation that is provided internally. Taking the USA as an example, numerous groups provide accreditation for internal healthcare organizations, including the Community Health Accreditation Program (CHAP), the Joint Commission, the Accreditation Commission for Health Care, Inc. (ACHC), the "Exemplary Provider Program" of The Compliance Team and the Healthcare Quality Association on Accreditation (HQAA).

Some other countries have looked towards accessing the services of the major international healthcare accreditation groups based in other countries to assess their healthcare services. There are many reasons for this, including cost, a desire to improve healthcare quality for one’s own citizens (good governance is at the basis of all high-quality healthcare), or a desire to market one’s healthcare services to “medical tourists”. Some hospitals go for international healthcare accreditation as a *de facto* form of advertising.

In response to this marketing opportunity, some national accreditation groups have expanded their wings internationally, and gone on to survey and accredit hospitals outside of their own national borders. When they choose to do this, such groups can be said to be providing "international healthcare accreditation".

This process of accreditation has been made increasingly complicated by the fact that in many parts of the world, more and more human beings are choosing to cross international borders to access healthcare, a phenomenon known as “medical tourism” or "Global Healthcare". Medical tourism/Global Healthcare cannot be ignored as a key issue in international healthcare accreditation - it is becoming increasingly important as millions of (especially) Europeans and Americans seek healthcare overseas outside of their own countries for a variety of reasons (including and especially affordability), and it represents a growing multi-billion euro/dollar/pound business of increasing importance to the economies of many countries, such as Singapore, Thailand, India, Hong Kong, Malaysia and the Philippines. The importance of medical tourism/Global Healthcare to the economy of developing countries is increasingly the subject of academic study. , and this synergy has a clear "knock-on" effect for those organizations based within the developed world who are seeking to develop the medical tourism/Global Healthcare market.

The reasons why patients are seeking out medical tourism/Global Healthcare options are manifold;

(a) healthcare may be too expensive at home
(b) waiting lists may be too long
(c) patients wish to access treatments not available at home (e.g. stem cell therapy, termination of pregnancy, unlicensed medications, gender re-assignment surgery)
(d) patients wish for greater confidentiality than may be feasible at home (e.g. HIV/AIDS treatment, infertility treatment, gender re-assignment surgery, face lifts)
(e) new challenges arise from time to time, such as new medical developments which are not universally accessible, the emergence of the so-called “superbugs” (e.g. MRSA, VRSA, VRE,

Clostridium difficile, ESBL-producing *E. coli*), problems with the blood transfusion supply (e.g. Chaga's disease in the USA, HIV, HTLV-1 etc.), and the social imponderables such as war, political change and natural disasters. Any of these factors may lead to a loss of public confidence in healthcare services, and a desire to seek out healthcare overseas. The environmental and political situation will constantly vary throughout the world, and this will need to be factored into the equations.

The following quotation, taken from the website of Partners Harvard Medical International, crystallizes the increasing relevance of international health care accreditation and its growing commercial importance, particularly in relation to medical tourism/global healthcare. "In competitive health care markets where patients have an increasing array of choices, quality is the most important differentiator for organizations striving for sustainability and both national and regional leadership. International accreditation, especially that granted by Joint Commission International (JCI), has become a powerful indicator of a health care organization’s commitment to high-quality care and patient safety." Reflecting this, much of the discussion on medical tourism blog sites reflects the increasing importance of international healthcare and hospital accreditation to this industry

Consumers

How does an individual contemplating becoming a medical tourist ensure that the overseas healthcare they are planning to access is as safe as possible and is of adequate quality? For sure, it is not simply a matter of looking at hospital buildings and at mattresses, and it is certainly not just an issue of looking only at the prices charged. While architecturally pleasing rooms and easier access to satellite television and the internet may improve personal comfort, and a bargain basement price may help the wallet, what is often more important may include such issues as:

- the standards of governance in the hospital or clinic
- the healthcare providing establishment's commitment to self-improvement, and to learn positively from errors
- the overall medical ethical standards operating within the organization
- the clinical staff's ethical standards and their personal and collective commitment to caring for patients and the wider community
- the quality of the clinical staff, including their background educational attainment and training, and evidence of continuing professional development by those staff
- the quality and ethical standards of the management and their personal and collective commitment to caring for patients and the wider community
- the clinical track record of the hospital or clinic
- the infection control track record of the hospital or clinic
- the hospital may be located in a country where the environment and climate may bring a patient into contact with infectious and/or tropical diseases that are unfamiliar to them
- evidence of a robust, just and fair system to deal with complaints made by patients when things go wrong, as they inevitably will from time to time, and where appropriate to compensate the injured party in a fair and reasonable way

The above list is not exhaustive, but it represents a good start. Also, the intending medical tourist should check whether or not a hospital is wholly accredited by an international accreditation group, or if it is only partly accredited (e.g. for infection control), the latter being less inclined to create confidence in a potential consumer.

How does the person in the street access this type of quality information? This can be very difficult. Accreditation schemes well-recognised as providing services in the international healthcare accreditation field include:

- Trent Accreditation Scheme (based in UK-Europe) The former Trent Scheme (which ended in 2010) was the first scheme to accredit a hospital in Asia, in Hong Kong in 2000 .
- QHA Trent Accreditation, based in the UK .

- Joint Commission International, or JCI (based in the USA) The first hospital to be accredited in Asia by JCI was Bumrungrad International Hospital, in 2002.
- Australian Council for Healthcare Standards International, or ACHSI (based in Australia)
- Accreditation Canada (formerly the Canadian Council on Health Services Accreditation or CCHSA)(based in Canada)
- Accreditation of France (La Haute Autorité de Santé) based in Paris, France.

The different accreditation schemes vary in approach, quality, size, intent and the skill of their marketing. They also vary in terms of costs incurred by hospitals and healthcare institutions. They all have web sites.

Umbrella organizations

The International Society for Quality in Health Care (ISQua) is an umbrella organisation for such organisations providing international healthcare accreditation . Its offices are based in the Republic of Ireland. ISQua is a small non-profit limited company with members in over 70 countries. ISQua works to provide services to guide health professionals, providers, researchers, agencies, policy makers and consumers, to achieve excellence in healthcare delivery to all people, and to continuously improve the quality and safety of care. ISQua does not actually survey or accredit hospitals or clinics itself.

The United Kingdom Accreditation Forum, or UKAF, is a UK-based umbrella organisation for organisations providing healthcare accreditation . Its offices are based in London. Like ISQua, UKAF does not actually survey and accredit hospitals itself.

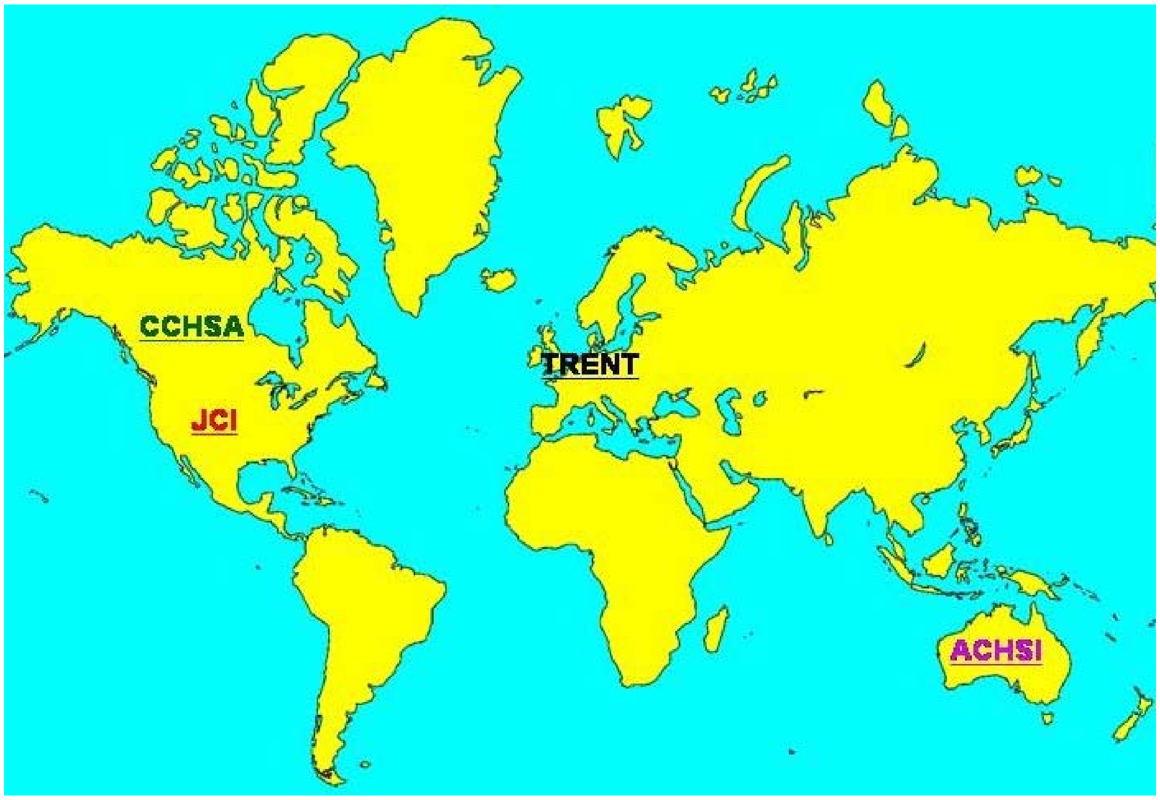
Accreditation services

If a hospital or clinic simply wishes to improve its services to patients wherever those patients come from (locally or from further afield), or wishes to attract medical tourists, how do they choose who to go to when contemplating accessing external peer review by an accreditation group such as those listed above.

There is a phrase in English – *"horses for courses"* – and no one healthcare system has a monopoly of excellence and no one provider country or scheme can claim to be the total arbiter of quality. The same is true of healthcare accreditation schemes.

For example, some countries (such as the USA) perform very poorly when it comes to providing anything close to universal access to healthcare of adequate quality to the population living within their own borders, while others (such as the UK) have tried to create state-funded systems which provide everything without the assistance of the private sector. Other differences exist – for example, general medical practice ("GP", also sometimes known as Family Medicine) is strong in the UK but weak in the USA, while US hospitals generally have greater expertise in marketing and billing. Different accreditation schemes are sourced out of different parts of the world, for example JCI out of the USA and North America, QHA Trent out of Europe, and ACHSI out of Oceania.

As no single international accreditation scheme enjoys exclusive rights to be seen as an overall world-wide-relevant scheme, some hospitals are looking towards multiple accreditation to achieve performance credibility in different parts of the world.



Locations of geographical bases of four of the world's leading international hospital accreditation schemes

With respect to the cost of accreditation, this can vary enormously and it can be hard to find out precise data; in the case of JCI, the costs can be substantial.

With respect to hospital work, ISO (the International Organization for Standardization) is often mistakenly considered to be an international healthcare accreditation scheme. It is not.

Chapter-11

EuResist, Quaternary Prevention and Pre- and Post-test Probability

EuResist

EuResist is an international project designed to improve the treatment of HIV patients by developing a computerized system that can recommend optimal treatment based on the patient's clinical and genomic data.

The project is part of the Virtual Physiological Human framework, funded by the European Commission. It started in 2006 with the formation of a consortium of several research institutes and hospitals in Europe and Israel. The consortium completed its commitment to the European Commission near the end of 2008, at which time the system became available online. A non-profit organization was consequently established by the main partners to maintain and improve the system.

In 2009, the EuResist project was named as a Computerworld honors program laureate.

Background

AIDS is a disease caused by the HIV retrovirus, which progressively reduces the effectiveness of the immune system, leading to infections and ultimately death.

More than 30 different drugs exist for treating HIV patients. Antiretroviral drugs can disrupt the virus's replication process causing its numbers to decrease dramatically. While the virus cannot be eradicated completely, in small numbers it is harmless. Usually a patient is given a combination of three or four drugs, a treatment known as highly active antiretroviral therapy, or HAART. The main reason such a treatment might fail is the development of mutated strands of the virus, resistant to one or more of the prescribed drugs.

Thus an important consideration when choosing treatment for a patient is to prescribe those drugs to which the particular patient's virus strands are most susceptible. One way to achieve that is to extract virus samples from the patient's blood and test them against all possible drugs. Since this process is lengthy and costly, computerized systems have been developed to predict virus resistance based on its genotype. The treating physician samples virus genotype sequences from the patient's blood and provides this data to a computerized system. The system then responds with drug recommendations.

Such systems are limited in accuracy, depending on the amount of data used for their creation, its quality and the richness of mathematical models used for the actual prediction. Prior to EuResist, such systems had several common characteristics that negatively impacted their accuracy:

- The amount of data used for creating the system was relatively small
- This data was in vitro data: laboratory measures of the resistance of various strands of the HIV virus to individual drugs. Such data is known to be inaccurate because laboratory tests do not simulate exactly the processes of a living organism, and since resistance to individual drugs does not accurately predict the resistance to a combination of drugs.
- They used a relatively simple mathematical prediction model

EuResist overview

EuResist sought to create a more accurate HIV treatment prediction system by collecting a large database of in vivo data (clinical and genomic records of real treatments of HIV patients and their consequences), and by using an array of prediction models instead of just one.

The database was created by merging local databases of various clinics across Europe. This database is thought to be the largest of its kind in the world. For each patient, it includes various personal and demographic details such as gender, age, country of origin, genomic sequencing of HIV viruses found in the patient's blood, records of the drugs prescribed, and the changes in the amount of virus in the blood following these treatments.

This data was used to train an array of prediction models, created by using various contemporary machine learning techniques, among them Bayesian networks, logistic regression, and others.

A web interface allows physicians to specify patients' clinical and genomic data. This data is sent to the prediction engines, and the combined response, which is displayed to the physician, includes various suggested treatments and a prediction of their effect on the amount of HIV virus in the blood.

The EuResist system was tested and compared with its predecessors by feeding it with historical data on patients for which treatment results are known. The developers of EuResist, who conducted this test, reported an improved performance over the previous state-of-the-art system.

History

EuResist started in 2006 as a consortium funded by the European Union as part of the Virtual Physiological Human FP-6 framework. The partners of this consortium were:

- Informa S.r.l. (Italy)
- University of Siena (Italy)
- Karolinska Institutet (Sweden)
- Universitätsklinikum Koeln (Germany)
- Max Planck Institute for Informatics (Germany)
- IBM Haifa Research Laboratory (Israel)
- MTA KFKI Reszecske-ES Magfizikai KutatoIntezet (Hungary)
- Kingston University (United Kingdom)

The consortium completed its commitment to the European Union in late 2008, at which time the EuResist system became available on line. The first five partners mentioned above continued to form a non-profit organization that maintains the system, expands the database with new clinical and genomical records and updates the prediction engines accordingly. As of mid-2010, an average of 600 queries are submitted to the EuResist system every quarter.

Recognition

On June 1, 2009, EuResist received a Computerworld honors program laureate award, a global program honoring individuals and organizations that use information technology to benefit society.

Quaternary prevention

The **quaternary prevention** are the action taken to identify patient at risk of overmedicalisation, to protect him from new medical invasion, and to suggest to him interventions, which are ethically acceptable.

Quaternary prevention is the set of health activities to mitigate or avoid the consequences of unnecessary or excessive intervention of the health system.

Social credit that legitimizes medical intervention may be damaged if doctors do not prevent unnecessary medical activity and its consequences. Quaternary prevention should take precedence over any alternative preventive, diagnostic and therapeutic, as is the practice version *primun non nocere*.

Concept

Prevention levels			**Doctor's side**	
			Disease	
			absent	**present**
Patient's side	**Illness**	**absent**	**Primary prevention** (illness absent disease absent)	**Secondary prevention** (illness absent disease present)
		present	**Quaternary prevention** (illness present disease absent)	**Tertiary prevention** (illness present disease present)

Main idea: to avoid patient overdiagnosis and overtreatment.

Use: During all the episode of care (preclinical and clinical period).

It's the "actions taken to identify patients at risk of overtreatment, to protect them from new medical procedures and ethically acceptable alternative to suggest". Concept coined by the Belgian general practitioner Marc Jamoulle and included in WONCA's *Dictionary of General and Family Medicine*.

To do quaternary prevention is to say “no” to many considerably indecent proposals, and to offer prudent and scientific alternatives (“ethics of negation” , “ethics of ignorance sharing”). To do quaternary prevention is to to exchange the fear exploited by healthcare malice for the feeling of knowing that what matters is the quality of life.

The intent of quaternary prevention is not to eliminate but rather to moderate the medicalization of the daily life, since a part of the aforementioned medicalization is not directly related to the medical intervention and has to do with social, cultural and psychologic reasons. Quaternary prevention is only about avoiding or palliating the medical part of the medicalization of the daily life.

To do quaternary prevention in clinical encounters is to comply with the scientific goal of Medicine, which aims for “the maximum quality with the minimum quantity, as close to the patient as possible”.

“To prevent is better than healing, when preventing is less harmful than healing”. To engage into quaternary prevention is to avoid the unnecessary curative and preventive activities. Every doctor-patient encounter should include quaternary prevention in order to avoid/limit the damage caused by the activity of the health system. To do it is to enforce the old motto primum non nocere.

Means

1.- **Narrative based Medicine**

The strongest means to accomplish this is to listen better to our patients. This is what has been termed Narrative based Medicine, which means to adapt the medically possible to the individual needs and wants. What we need is a strong and sustainable relationship with our patients and their trust in our honesty and specific knowledge.

2.- **Evidence-based medicine**

The other important means is called Evidence based Medicine. The knowledge of the probable predictive values of diagnostic tests and the probabilities of effect sizes of benefit and harm of therapy and preventive measures give us the opportunity to leave out many useless procedures.

Intervention types

Healthcare professionals must be aware of the consequences of their decisions, and include quaternary prevention interventions in their daily clinical practice with each patient.

- To prevent the cascade effect:

  - To prevent the diagnostic cascade
  - To prevent the therapeutical cascade

- To prevent disease promotion
- To prevent medicalization

Activities

- Do not mistake risk factor with disease.
- To avoid check ups or unecessary exams.
- To avoid technical interventionism in healthcare.
- To avoid scoliosis.
- To avoid the pharmacological treatment of hypercholesterolemia in primary prevention.
- To avoid hormone replacement therapy during menopause
- To avoid the indiscriminate use of antibiotics (very often unnecessary, with the subsequent unjustified increase of bacterial resistances)
- To avoid unnecessary diagnosis of genetic disorders (for example: the promotion of the haemocromatosis screening, of doubtful scientific value, but of undoubtful effect in terms of the medicalization of society.
- To avoid the overdiagnosis and overtreatment of the attention deficit hyperactivity disorder (ADHD).

- To ellaborate validated diagnostic and therapeutic protocols that are effective in the prevention of the renal injury in the minority of patients with “complicated” pielic ectasia, and that avoid the excessive amount of intervention in the majority of the patients with “simple” pielic ectasia.

Pre- and post-test probability

Pre-test probability and **post-test probability** are the probabilities of a having condition before and after a diagnostic test. *Post-test probability*, in turn, can be *positive* or *negative*, depending on whether the test falls out as a positive test or a negative test, respectively.

Although sometimes used synonymously with *positive* and *negative predictive value*, *positive* and *negative post-test probability* generally refer to probabilities of individuals, while predictive values refer to those established by control groups. Still, if the individual's pre-test probability of the target condition is the same as the prevalence in the control group used to establish the positive predictive value, then the predictive values and post-test probabilities are numerically equal.

Calculation of post-test probability

If pre-test probability is equal to population prevalence

If assuming that there are no other known risk factors in the individual being tested that would result in another pre-test probability than the general population, then the *pre-test probability* (the probability of a having a target condition before a diagnostic test) can be regarded as being numerically equal to the prevalence of the condition in the population.

In such cases, pre-test probability can be calculated from the diagram just below as follows (with dashes removed to avoid mixup with the minus sign):

Pretest probability = (True positive + False negative) / Total sample

		Condition (as determined by "Gold standard")		
		Positive	***Negative***	
Test outcome	***Positive***	**True Positive**	**False Positive** (Type I error)	→ Positive predictive value
	Negative	**False Negative** (Type II error)	**True Negative**	→ Negative predictive value

	↓ Sensitivity	↓ Specificity	

Also, with the same assumption, the *positive post-test probability* (the probability of having the target condition if the test falls out positive), is numerically equal to the positive predictive value, and the *negative post-test probability* (the probability of having the target condition if the test falls out negative) is numerically equal to the negative predictive value, again assuming that the individual being tested does not have any other risk factors that result in that individual having a different *pre-test probability* than the control group used to establish the positive and negative predictive values of the test.

In the diagram above, this *positive post-test probability* is calculated as (with dashes removed to avoid mixup with the minus sign):

Posttest probability = True positive / (True positive + False positive)

If pre-test probability is different from population prevalence

The above methods are inappropriate to use in an individual being tested that has a different pre-test probability than the general population, or more specifically, having a different pre-test probability than the average one in control group of people used to establish the positive predictive value of the test. Examples of causes of such differences include known risk factors of the individual that were not generally present in the control group, or previous tests, physical examination or medical history on the individual that have conferred a different risk profile.

As a result, the *prevalence* in the population is not completely accurate in representing the *pre-test probability* of the individual, and the *predictive value* (whether *positive* or *negative*) is not completely accurate in representing the *positive pre-test probability* of the individual of having the target condition.

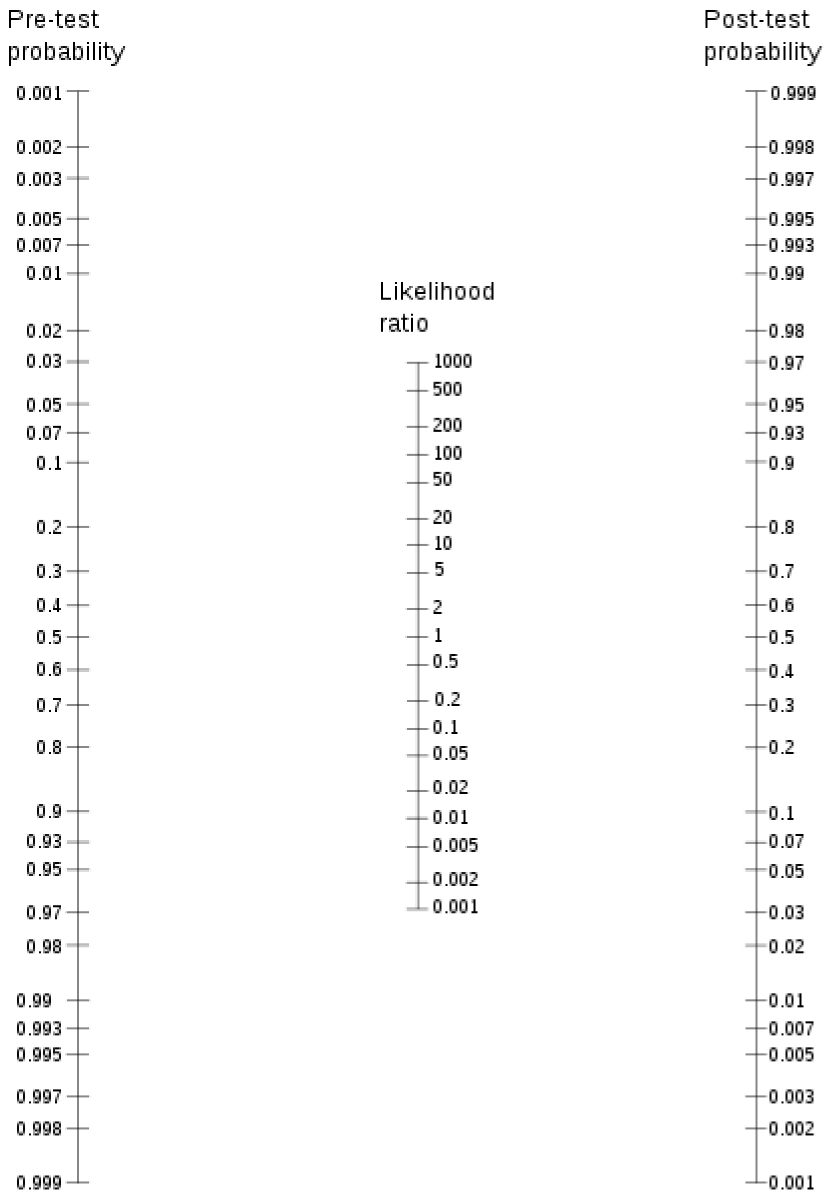
Still, if the pre-test probability of the individual is known, then a *post-test probability* can be estimated using likelihood ratio. *Likelihood ratio* is calculated from sensitivity and specificity of the test, and does not depend on prevalence in the population. A thought experiment is to imagine that individual as belonging to a group with increased or decreased prevalence, such as, theoretically, a group of people with exactly the same risk factors as the individual at hand, and the prevalence in this group would be different from that of a control group taken from the general population.

Estimation of post-test probability from pre-test probability and likelihood ratio goes as follows:

- Pretest odds = (Pretest probability / (1 - Pretest probability)
- Posttest odds = Pretest odds * Likelihood ratio

In equation above, *positive post-test probability* is calculated using the *likelihood ratio positive*, and the *negative post-test probability* is calculated using the *likelihood ratio negative*.

- Posttest probability = Posttest odds / (Posttest odds + 1)



Fagan nomogram

The relation can also be estimated by a so called *Fagan nomogram* (shown at right) by making a straight line from the point of the given *pre-test probability* to the given *likelihood ratio* in their scales, which, in turn, estimates the *post-test probability* at the point where that straight line crosses its scale.

The post-test probability can, in turn, be used as pre-test probability for additional tests if it continues to be calculated in the same manner.

However, *post-test probability*, as estimated from the *likelihood ratio* and *pre-test probability*, should still be handled with caution in individuals with other risk factors than the general population, because such factors may also influence the test itself in unpredictive ways, still causing inaccurate results. Preferably, a large control group of equivalent individuals should be studied, in order to establish separate *positive* and *negative predictive values* for use of the test in such individuals.

Example

An individual was screened with the test of fecal occult blood (FOB) to estimate the probability for that person having the target condition of bowel cancer, and it fell out positive (blood were detected in stool). Also, that individual happens to have heredity for bowel cancer, and let's say that this heredity in itself doubles the risk of cancer for this individual.

The sensitivity, specificity etc. of the FOB test were established with a control group of 203 people (without such heredity), and fell out as follows:

		Patients with bowel cancer (as confirmed on endoscopy)		
		Positive	***Negative***	
Fecal occult blood screen test outcome	***Positive***	**TP = 2**	**FP = 18**	→ Positive predictive value = TP / (TP + FP) = 2 / (2 + 18) = 2 / 20 **= 10%**
	Negative	**FN = 1**	**TN = 182**	→ Negative predictive value = TN / (FN + TN) = 182 / (1 + 182) = 182 / 183 **≈ 99.5%**
		↓ Sensitivity = TP / (TP + FN) = 2 / (2 + 1) = 2 / 3 **≈ 66.67%**	↓ Specificity = TN / (FP + TN) = 182 / (18 + 182) = 182 / 200 **= 91%**	

From this, the *likelihood ratios* of the test can be established:

1. Likelihood ratio positive = sensitivity / (1 − specificity) = 66.67% / (1 − 91%) = 7.4
2. Likelihood ratio negative = (1 − sensitivity) / specificity = (1 − 66.67%) / 91% = 0.37

In this example, the pre-test probability of the individual is assumed to be twice that of the prevalence of bowel cancer in the control group used in this study, as that prevalence may be assumed to represent that of the general population:

- Prevalence = (2 + 1) / 203 = 0.0148
- Pretest probability = 0.0148 * 2 = 0.0296
- Pretest odds = 0.0296 / (1 - 0.0296) = 0.0305
- Positive posttest odds = 0.0305 * 7.4 = 0.226
- Positive posttest probability = 0.226 / (0.226 + 1) = 0.184 or 18.4%

Thus, that individual has a risk of 18.4% of having bowel cancer. This is almost (but not fully) twice the equivalent measure in the control group.

However, that individual's heredity may possibly interfere with the FOB test in unpredictive ways, such as also giving an increased bleeding tendency that would increase the false positive probability, making the result in this example invalid. Ideally, separate *positive* and *negative predictive values* should therefore be established by a large group of individuals that have the same hereditary predisposition.