A Research Strategy on Techniques of Software Testing and Quality of Service Testing Methodology

Neha Singh & Pooja Yadav
Department of Computer Science, Banarsidas Chandiwala Institute of Professional Studies

Abstract

Most important concern for software development in recent time is developing unswerving and robust software. Software testing is as old as the hills in the history of digital computers. The testing of software is an important means of assessing the software to determine its quality. Since testing typically consumes 40–50% of development effort, it consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering. With the development of Fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made [4]. Despite advances in formal methods and verification techniques, a system still needs to be tested before it is used. Testing remains the truly effective means of assure the quality of a software system of non-trivial complexity [4], as well as one of the most intricate and least understood areas in software engineering [6]. Testing, an important research area within computer science is likely to become even more important in the future.

This retrospective on a fifty-year of software testing technique research examines the maturation of the software testing technique research by tracing the major research results that have contributed to the growth of this area. It also assesses the change of research paradigms over time by tracing the types of research questions and strategies used at various stages. To study the effects of QoS solutions/techniques on the traffic, methods for testing need to be developed. An assumption has developed over the last decade that the Internet can support any form of communication service. The services range from traditional reliable, non-real time data transfer from one computer to the other, to time critical services such as audio and video streaming. These services require the network to negotiate certain performance attributes between end applications and the core. Quality of Service (QoS) as the name suggests aims to provide performance attributes in the form of service guarantees that providers can offer to their customers. Network routers have had QoS implementations for some time now, but there have been no universally agreed upon metric or methodology with which to test QoS. This paper proposes a testing methodology for gathering and analysing relevant QoS data and, if possible, suggesting changes to improve performance.

II. The Taxonomy of Testing Techniques

Software testing is a very broad area, which involves many other technical and non-technical areas, such as specification, design and implementation, maintenance, process and management issues in software engineering. Our study focuses on the state of the art in testing techniques, as well as the latest techniques which representing the future direction of this area. Before stepping into any detail of the maturation study of these techniques, let us have a brief look at some technical concepts that are relative to our research.
A. The Goal of Testing

In different publications, the definition of testing varies according to the purpose, process, and level of testing described. The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances. Miller’s description of testing views most software quality assurances activities as testing. He contends that testing should have the major intent of finding errors. A good test is one that has a high probability of finding an as yet undiscovered error, and a successful test is one that uncovers an as yet undiscovered error. [7]

B. The Testing Spectrum

Testing is involved in every stage of software life cycle, but the testing done at each level of software development is different in nature and has different objectives.

Unit Testing is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software, and is often called “unit”, “module”, or “component” interchangeably.

Integration Testing is performed when two or more tested units are combined into a larger structure. The test is often done on both the interfaces between the components and the larger structure being constructed, if its quality property cannot be assessed from its components.

System Testing tends to affirm the end-to-end quality of the entire system. System test is often based on the functional/requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability, are also checked.

Acceptance Testing is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors. [7]

C. Static Analysis and Dynamic Analysis

Based on whether the actual execution of software under evaluation is needed or not, there are two major categories of quality assurance activities:

Static Analysis focuses on the range of methods that are used to determine or estimate software quality without reference to actual executions. Techniques in this area include code inspection, program analysis, symbolic analysis, and model checking.

Dynamic Analysis deals with specific methods for ascertaining and/or approximating software quality through actual executions, i.e., with real data and under real (or simulated) circumstances. Techniques in this area include synthesis of inputs, the use of structurally dictated testing procedures, and the automation of testing environment generation. Generally the static and dynamic methods are sometimes inseparable, but can almost always discussed separately. In this paper, we mean dynamic analysis when we say testing, since most of the testing activities (thus all the techniques studied in this paper) require the execution of the software. [7]

D. Functional Technique and Structural Technique

The information flow of testing is shown in Figure 1. As we can see, testing involves the configuration of proper inputs, execution of the software over the input, and the analysis of the output. The “Software Configuration” includes requirements specification, design specification, source code, and so on. The “Test Configuration” includes test cases, test plan and procedures, and testing tools. Based on the testing information flow, a testing technique specifies the strategy used in testing to select input test cases and analyze test results. Different techniques reveal different quality aspects of a software system, and there are two major categories of testing techniques, functional and structural.

Functional Testing: the software program or system under test is viewed as a “black box”. The selection of test cases for functional testing is based on the requirement or design specification of the software entity under test. Examples of expected results, sometimes are called test oracles, include requirement/design specifications, hand calculated values, and simulated results. Functional testing emphasizes on the external behavior of the software entity. Structural Testing: the software entity is viewed as a “white box”. The selection of test cases is based on the implementation of the software entity. The goal of selecting such test cases is to cause the execution of specific spots in the software entity, such as specific statements, program branches or paths. The expected results are evaluated on a set of coverage criteria. Examples of coverage criteria include path coverage, branch coverage, and data-flow coverage. Structural testing emphasizes on the internal structure of the software entity.

III. Scope of the Study

A. Technical Scope

In this paper, we focus on the technology maturation of testing techniques, including these functional and structural techniques that have been influential in the academic world and widely used in practice. We are going to examine the growth and propagation of the most established strategy and methodology used to select test cases and analyze test results. Research in software testing techniques can be roughly divided into two branches: theoretical and methodological, and the growth in both branches push the growth of testing technology together. Inhibitors of maturation, which explains why the in-depth research hasn’t brought revolutionary advantage in industry testing practice, are also within our scope of interest. There are many other interesting areas in software testing. We limit the scope of our study within the range of testing techniques, although some of the areas maybe inseparable from our study. Specifically, we are not going to discuss: [7]

- How testing is involved in the software development cycle
- How different levels of testing are performed
B. Goal and standard of progress

The ultimate goal of software testing is to help designers, developers, and managers construct systems with high quality. Thus research and development on testing aim at efficiently performing effective testing – to find more errors in requirement, design and implementation, and to increase confidence that the software has various qualities. Testing technique research leads to the destination of practical testing methods and tools. Progress toward this destination requires fundamental research, and the creation, refinement, extension, and popularization of better methods. The standard of progress for the research of testing techniques includes:

- Degree of acceptance of the technology inside and outside the research community
- Degree of dependability on other areas of software engineering
- Change of research paradigms in response to the maturation of software development technologies
- Feasibility of techniques being used in a widespread practical scope, and Spread of technology – classes, trainings, management attention

C. What is QoS?

There is no common or formal definition of QoS. However, there are a number of definitions at the communication level where the notion originated to describe technical characteristics of mainly non-time-dependent data transmission. Emerging networks such as ATM, can provide QoS guarantees on bandwidth and delay for the transfer of continuous media (CM) data.

- The International Telecommunication Union (ITU) standard X.902, *Information technology - Open distributed processing - Reference Model*, refers to QoS as “A set of quality requirements on the collective behavior of one or more objects.” A number of QoS parameters describe the speed and reliability of data transmission, e.g., throughput, transit delay, and error rate.

- The *ATM Lexicon* defines QoS as "A term which refers to the set of ATM performance parameters that characterize the traffic over a given virtual connection." QoS parameters apply mostly to lower level protocol layers, and were not meant to be directly observable or verifiable by the application. These parameters include cell loss ratio, cell error rate, cell misinsertion rate, cell delay variation, cell transfer delay, and average cell transfer delay. **Five service classes** are defined in terms of QoS parameters. Class 0 refers to best effort service in which no specific traffic parameters and no absolute guarantees are provided.

- Recently the Internet Engineering Task Force (IETF) has begun to address QoS issues for ATM. RFC 1946, *Native ATM Support for ST2+*, states "As the demand for networked real time services grows, so does the need for shared networks to provide deterministic delivery services. Such deterministic delivery services demand that both the source application and the network infrastructure have capabilities to request, setup, and enforce the delivery of the data. Collectively these services are referred to as bandwidth reservation and Quality of Service (QoS)." RFC 1932, *IP over ATM: A Framework Document*, states "QoS parameters [for real-time applications] are assumed to precede traffic in RSVP or be carried in some form within the traffic itself." "Work in progress is addressing how QoS requirements might be expressed and how the local decisions might be made as to whether those requirements are best and/or most cost effectively accomplished using ATM or IP capabilities." RSVP is discussed further below.

The IEEEE paper, *Distributed Multimedia and Quality of Service: A Survey*, provides a more general definition of QoS for applications that must communicate in real-time: "The set of those quantitative and qualitative characteristics of a distributed multimedia system, which are necessary in order to achieve the required functionality of an application." This paper also provides a model of QoS processing for multimedia systems which we will generalize below to include applications requiring Internet services. Several research groups are investigating QoS support for the WWW, in particular, researchers at BBN, the Distributed Systems Technology Centre, and Washington University. These and other QoS-based projects are summarized below. [1]

IV. Quality of Service

Quality of Service can be thought of as a mechanism to satisfy a level of service performance across a network. It can also be understood as the ability of a network to provide desired handling of the traffic such that the network meets the level of service that end applications expect.

A. Early days

In the early iterations of the Internet protocol (IP), getting a packet to its destination was of primary concern. Internet Service Providers (ISP) were more involved in keeping up with the increasing demands of the customers and keeping the network up and running. IP's initial design included a Type of Service (TOS) "eld intended for offering different categories of treatment. Using this feature, however, was not a priority for service providers. At the same time, over-engineering hampered the deployment of QoS; a network was always provisioned for more traffic than it actually had to handle to avoid congestion and contention for resources. As contention of resources is one of main reason for QoS to be deployed, it was avoided by over engineering. Another, relatively minor factor in the slow growth of QoS was the complexity of billing systems. Originally, providers had fairly simple billing systems, based on just time and access capacity. Migrating to a billing system smart enough to differentiate between varied service plans and customers was seen as prohibitively difficult, given that it might require the system to look into 'almost' packet level granularity. [
How to deliver QoS?

A QoS-enabled network must be able to handle different traffic streams in different ways. This necessitates categorizing traffic into types or classes and defining how each type is handled. All the following aspects could be considered within the scope of QoS [1]:

- Differentiation of traffic (Classification)
- Admission control
- Congestion management

1 Differentiation of Traffic

IP routers are designed to treat all packets in the same way. Packets arriving at the router are considered as new event without having any local state memory. So the basic building block here is the ability to classify the traffic into different categories based on what service they are to receive. Once the packets have been classified they can be given differential treatment based on the classification. One of the easiest ways to classify a packet is to embed the packet header with an information field that specifies the class of traffic the packet belongs to. This way of classification is known as packet service classification. The advantage of this method is that the packet need not be classified at any router in the network; instead appropriate treatment can be given to the packet based on the service classification field. Differentiated Services architecture [3] could be considered as the IETF's initiative towards standardizing this process. This method can be problematic if an excessive number of applications are sending packets expecting premium handling. Taking in packets and assigning them specific behavior based on a value set in the packet header can tax the router's resources. Another stumbling block is that different networks may use different codes to signify the same category of service. In this case, the TOS or differentiated service field might be remarked before being admitted to a network. This marking or remarking is often involved in controlling congestion. The classification of a packet into a service class can be done based on the following parameters [1]:

- Service Mark
- Protocol
- Destination protocol port
- Source protocol port
- Destination host address
- Source host address
- Source device interface
- Any combination of above

2 Admission Control

When a packet arrives at the ingress of a network, it can be classified into two categories, namely in profile and out of profile. If a packet conforms to an agreement with the owner (service provider) of the network while entering, it will be classified as in profile and will be allowed to enter the network with or without any changes to the packet. For example, the ingress router can be configured to allow only TCP packets to go through. In this case any TCP packet arriving at the ingress router is classified as in profile and all other are considered out of profile. But if the packet does not conform then it can be dropped at the entry point into the network. The ingress router can be configured to utilize any of the parameters listed above for classification to determine whether the packet is in or out of profile. Sometimes packets may be dropped based on the scarcity of the resources available to handle them, regardless of whether a packet is in profile or out. There are a few ways in which a device can judge the available resources of the network.

3 Congestion Management

Congestion management can be defined as the ability of a network to effectively deal with heavy traffic volumes [2]. This aspect of QoS is more than just a single entity and mostly encompasses more than one tool to make it work. For example, differential congestion management is when a lower-precedence class has a higher probability of being dropped and additional buffer space is only available to traffic of higher class. Queuing can be one of the tools which implement congestion management for the router and the network. For example, additional congestion management techniques can be combined with weighted queuing to give higher preference to higher class of traffic to create a congestion management solution.

V. Generalized QoS Processing Model

To build QoS into a system involves

- the construction of a generalized QoS framework,
- QoS specification which captures application QoS requirements,
- mapping of QoS requirements to resources,
- QoS mechanisms which realize desired QoS behavior.

A. QoS Specification

An application's QoS requirements are conveyed in terms of high-level parameters that specify what the user requires. QoS specification is different at each system layer and is used to configure QoS mechanisms at each layer. Possible system layers are

- protocols - transport, network
network
middleware
operating system - scheduling, resource management, real-time support
distributed platforms - CPU, memory/buffers, devices
application

QoS specification encompasses requirements for:
- performance - expected performance characteristics are needed to establish resource commitments,
- synchronization - characterizes the degree of synchronization required between related services, events, or information flows,
- level of service - specifies the degree of resource commitment required to maintain performance guarantees,
- cost of service - the price a user is willing to incur to obtain a level of service,
- QoS management - the degree of QoS adaptation that can be tolerated and scaling actions to be taken in the event the contracted QoS cannot be met.

QoS requirements are assessed to determine if they can possibly be met. If, for example, the level of service requested cannot be provided, the user can be asked if a certain level of degradation is acceptable before proceeding further.

B. QoS Mapping

QoS requirements are used to derive resource requirements for entities such as computation, communication, and storage. They are successively mapped into quantitative QoS parameters relevant to various system layers that can be monitored and controlled. QoS parameters may be oriented towards:
- performance - sequential versus parallel processing, delays, data rate
- format - transfer rate, data format, compression schema, image resolution
- synchronization - loosely versus tightly coupled, synchronous versus asynchronous
- cost - platform rates, copyright fees, connection and data transmission rates
- user - subjective quality of images, sound, response time

Each QoS parameter can be viewed as a typed variable with bounded values, and the values are subject to negotiation between the system layers.

C. QoS Enforcement

To provide and sustain QoS, resource management must be QoS-driven. In allocating resources, the resource management system must not only consider resource availability and resource control policies, but also an application’s QoS requirements measured in terms of the QoS parameters. To ensure the contracted QoS is sustained, it must monitor QoS parameters and reallocate resources in response to system anomalies. Prior to allocating resources, the system layers negotiate to determine if they can collectively ensure that the required QoS parameters can be consistently satisfied. Negotiation involves dynamic adaptation and the transmission and translation of QoS parameters between the layers as the layers enter into different types of agreements, e.g., guaranteed, best-effort, or predictive. If negotiation ends in agreement, the application is launched. After resources are allocated, QoS mechanisms at each layer guarantee the contracted QoS, and the resource manager guarantees the sustained availability of the allocated resources. This requires monitoring resource availability and its dynamic characteristics, e.g., measuring processing workload and network traffic, to detect deviations in the QoS parameters. When there is a change of state, i.e., degradation in the QoS, and the resource manager cannot make resource adjustments to compensate (e.g., reschedule shared resources to satisfy allocations or switch to an optimized implementation of an object/service), then the application is notified, e.g., application handlers are called. The application can either adapt to the new level of QoS or scale to a reduced level of service. [1]

VI. Conclusions

Testing has been widely used as a way to help engineers develop high-quality systems, and the techniques for testing have evolved from an ad hoc activity means of small group of programmers to an organized discipline in software engineering. However, the maturation of testing techniques has been fruitful, but not adequate. Pressure to produce higher-quality software at lower cost is increasing and existing techniques used in practice are not sufficient for this purpose. Fundamental research that addresses the challenging problems, development of methods and tools, and empirical studies should be carried out so that we can expect significant improvement in the way we test software. Researchers should demonstrate the effectiveness of many existing techniques for large industrial software, thus facilitating transfer of these techniques to practice. The successful use of these techniques in industrial software development will validate the results of the research and drive future research. The pervasive use of software and the increased cost of validating it will motivate the creation of partnerships between industry and researchers to develop new techniques and facilitate their transfer to practice. Development of efficient testing techniques and tools that will assist in the creation of high-quality software will become one of the most important research areas in the near future.

Quality of Service testing methodology should effectively test an IP network implementation. Testing multiple implementations helps us refine the methodology and also helps in identifying problem areas in an implementation.

VII. Bibliography

[3] D. Gelperin and B. Hetzel, “The Growth of Software Testing”, Communications of the ACM, Volume 31 Issue 6, June 1988, pp. 687-695 [GH88] In this article, the evolution of software test engineering is traced by examining changes in the testing process model and the level of professionalism over the years. Two phase models, the demonstration and destruction models, and two life cycle...
models, the evolution and prevention models are given to characterize the growth of software testing with time. Based on the models a prevention oriented testing technology is introduced and analyzed in detail.


[5] E. F. Miller, “Introduction to Software Testing Technology,” Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16 [Miller81] This article serves as the one of the introductory sections of the book Tutorial: Software Testing & Validation Techniques. A cross section of program testing technology before and around the year 1980 is provided in this book, including the theoretical foundations of testing, tools and techniques for static analysis and dynamic analysis, effectiveness assessment, management and planning, and research and development of software testing and validation. The article briefly summarizes each of the major sections. The article also gives good view of the motivation forces, the philosophy and principles of testing, and the relation of testing to software engineering.


