RESTful Web Service Protocol Testing Using TTCN3

Dissertation submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Prabhjot Singh Bal

Registration number

41400003

Supervisor

Dalwinder Singh

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month…May… Year …2017…

PAC COMMITTEE MEMBERS APPROVAL

DECLARATION STATEMENT

I hereby declare that the research work reported in the dissertation entitled "RESTFUL WEB SERVICE PROTOCOL TESTING USING TTCN-3” in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab, is an authentic work carried out under supervision of my research supervisor Prof. Dalwinder Singh. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University’s Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

Signature of Candidate

Prabhjot Singh Bal

R.No 41400003

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation entitled "RESTFUL WEB SERVICE PROTOCOL TESTING USING TTCN-3**"**, submitted by **Prabhjot Singh Bal** at **Lovely Professional University, Phagwara, India** is a bonafide record of his / her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree.

Signature of Supervisor
(Prof. Dalwinder Singh)

Date:

Counter Signed by:

1) **Concerned HOD:**
HoD's Signature: ________________

HoD Name: ____________________

Date: ___________________

2) **Neutral Examiners:**

External Examiner

Signature: _______________

Name: __________________

Affiliation: ______________

Date: ___________________

Internal Examiner

Signature: _______________

Name: __________________

Date: ___________________

ACKNOWLEDGEMENT

This Master of Technology Dissertation work is done for fulfilling the requirements of the **School of Computer Science and Engineering, Lovely Professional University, Phagwara, Punjab,** during the period Jan 2017 to June 2017.

I would like to thank my supervisor **Prof. Dalwinder Singh** mostly and also other Computer Science Department faculty members for showing interest in this dissertation work, and providing frequent guidance and pointing out flaws in preparing, planning, research methodology and presentation. Without the support of my advisor this dissertation work would not have been possible, as he kept me on the right track of doing the right work at the right time as required.

I am very grateful to Prof. Dalwinder Singh for showing extreme patience and inspiring persistence while guiding me through this dissertation work.

Phagwara, May 29th, 2017

Prabhjot Singh Bal

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS / ABBREVIATIONS

No.	ABBREVIATION	DESCRIPTION
1.	TTCN-3	Testing and Test Control Notation 3
2.	SOAP	Simple Object Access Protocol
3.	DTD	Document Type Definitions
4.	REST	Representational State Transfer
5.	SUT	System Under Test
6.	UML	Unified Modelling Language
7.	SOA	Service Oriented Architecture
8.	ETSI	European Telecom Standardization Institute
9.	XML	Extensible Markup Language
10.	JSON	JavaScript Object Notation
11.	IDE	Integrated Development Environment
12.	TM	TCN-3 Test Management
13	TCI	TTCN-3 Test Control Interface
14	TRI	TTCN-3 Test Runtime Interface
15	MSC	Message Sequence Chart

ABSTRACT

Web Services are considered an essential services-oriented technology on networked application architectures due to their language and platform-independence. Their language and platform independence also brings difficulties in testing these services. In this dissertation, a comparative evaluation of testing techniques based on, TTCN-3 and SoapUI, for contributing towards resolving these difficulties is performed. This dissertation's approach distributes test activities on both server and client sides to ensure a well-organized web testing process. By specifying test suites at an abstract level in TTCN-3 as against GUI commands of SoapUI they are programming language and platform independent, and can be reused with testing of multiple Web Services on the Internet and Intra-nets. The major communication protocol used in recent past for Web Services was SOAP being mainly XML over HTTP, but more recent trends brought RESTful Web Services, and Web-Socket enabled Services etc. Web Service testing considers functionality, load and stress aspects to measure how a Web Service performs for single clients and scales as the number of clients accessing it increases. This dissertation work explores the automated testing of Web Services by using both TTCN-3 and SoapUI. A comparative evaluation technique is attempted for server and client side testing processes mapping RESTful service descriptions to TTCN-3 Abstract Test Suite on server end, and RESTful client test systems with both TTCN-3 adapters and SoapUI API, with encoding components implemented for each. In this dissertation, it is explored how a test specification and implementation approach with testing tools like TTCN-3 and SoapUI can be used to define test suites for RESTful Web Services employing different levels of abstraction and implementation. Aspects of TTCN-3 and SoapUI are demonstrated, including test execution and a powerful mechanisms in TTCN-3 that allows a separation between behavior and the conditions governing behavior.

CHAPTER 1
INTRODUCTION

1.1 Software Testing

Software testing is the process of validating and verifying a software program or application. It involves checking for the correct functionality of the system. It also confirms that a project meets the technical and business requirements as expected. The testing procedure includes measurement and comparison of the expected outcome with the actual outcome, which helps in detecting the errors which might occur during the analysis, design and development of the project.

Any software application must not result in failures and give correct results. In order to ensure this, defects in the software need to be identified at the earliest stages of the software life cycle. These can become very expensive in the future or the later stages of the software development life cycle. Hence timely testing can ensure the quality of the software product and in turn when delivered to the customers helps in gaining their confidence.

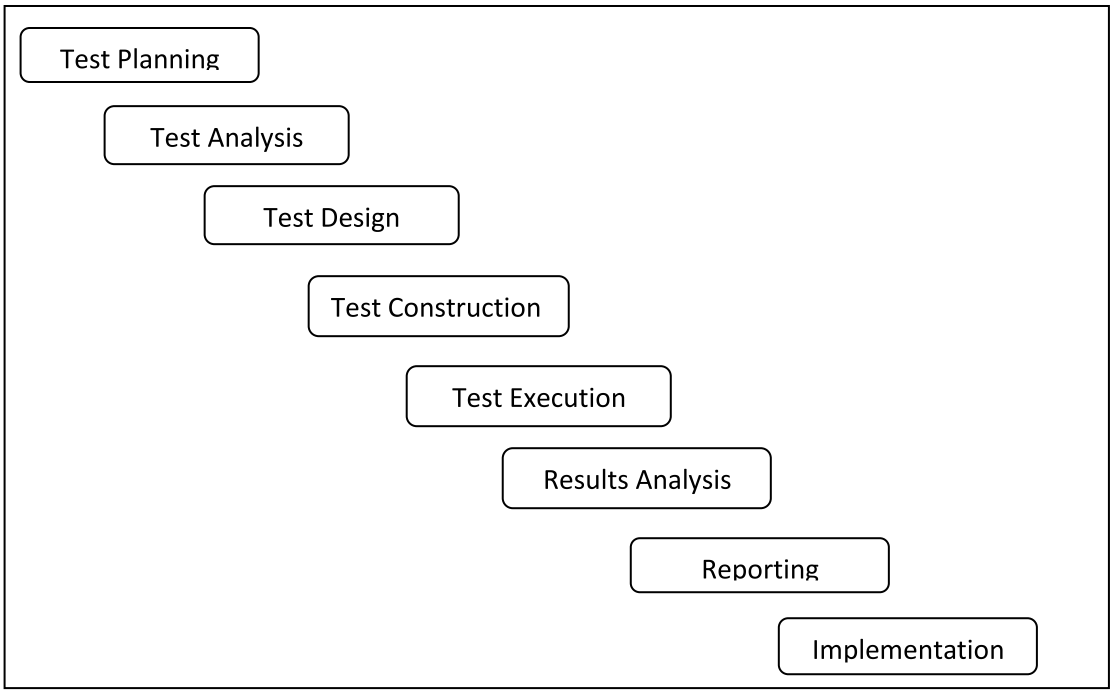


Figure 1.1: Software Testing Life Cycle

Two important terms needed to be clear about when considering software testing are:

Validation: Software validation is a dynamic process. This method is used to ensure that software satisfies all the requirements that were laid out for its construction. This validation is usually done both during and at the end of the software development life cycle.

Verification: Software verification is a static process. Verification tests those very conditions which were laid out at the starting of the development cycle in order to confirm whether they have been satisfied or not.

1.2 Software Testing Techniques

Software Testing can be classified into the following forms:

1. Black Box Testing and White Box Testing.
2. Manual Testing and Automated Testing.
3. Static Testing and Dynamic Testing.

1.2.1 Black Box Testing

Black Box method is a type of testing in which the test developer and executor need not have any knowledge about the internal workings of the software. It is also referred to as Functional Testing. The Tester develops test cases on the basis of functional specifications and interfaces of the software. It does not require any knowledge of the implementation and it can be done along with development.

Input
Output

Figure 1.2: Black-Box Testing

Types of Black Box Testing:

Boundary Value Analysis: In this type of testing different boundary conditions are tested. Both invalid and valid boundaries are identified first. On the basis of these boundary conditions, the test cases are developed and executed against the software under test. It is very useful and easy to apply and provides good results in terms of fault detection.

Equivalence Partitioning: In this method the software under test is divided into various partitions. The division of the partitions is on the basis of similarity of data usage or similar functionality. Valid and Invalid test cases are developed to test these same conditions. The invalid conditions should give wrong results whereas the valid conditions should provide the correct result.

State Transitions: State transitions testing is mainly conducted to test the transitional changes in the system under test. When an action is taken to cause change of system state from one to another, pre and post conditions have to be tested. It also requires a graphical presentation to ensure consistency of state changes.

1.2.2 White Box Testing

White Box testing is also called as the structural or code based testing. In this type of testing, the tester has good knowledge of the internal workings of the system under test. It is also known as clear glass testing, in which all internal code, control, conditional branches of code are known to the tester. The test cases are executed to validate the code. It is usually done at the Unit or Integration testing level.

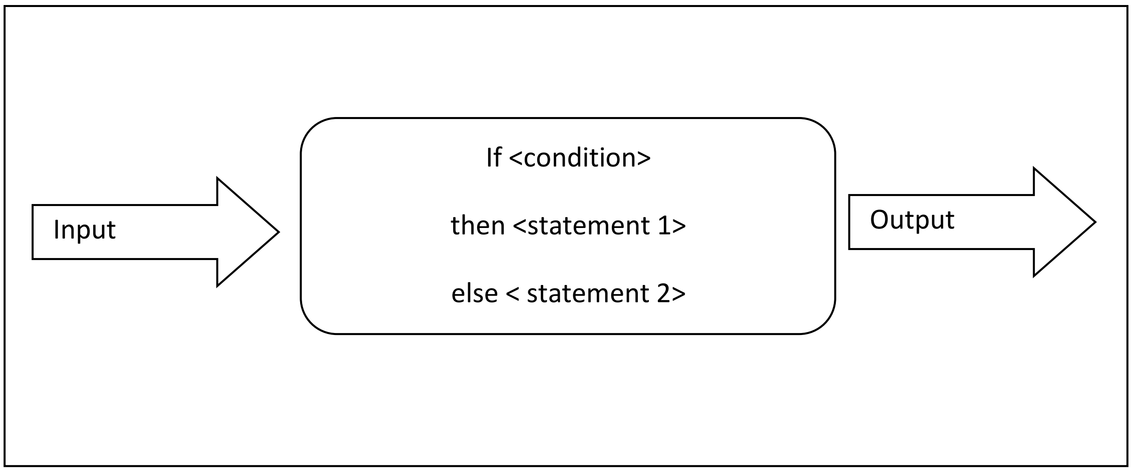


Figure 1.3: White-Box Testing

Different types of White Box testing techniques are:

Statement Coverage: In this type of testing the test cases are built to cover all possible statements of code. This type of testing is usually done during Unit testing. The aim is to cover each and every statement of code in the software under test.

Branch Coverage: In this type of testing the goal is to ensure that each one of the possible branches from each decision point is executed at-least once and thereby ensuring that all reachable code is executed correctly.

Condition Coverage: It is to test all the conditions of the system under test. Test cases are developed to test both valid and invalid conditions. Condition coverage is also known as Predicate Coverage in which each one of the Boolean expressions have to be evaluated to both True and False values.

Path Coverage: In this type of testing all the paths leading to the final decisions are tested. All possible control paths are tested, including all loop paths. The test cases are prepared based on the logical complexity measurement of a software design.

1.2.3 Manual and Automated Testing

In manual testing the system under test is tested manually, which means to say without using any automatic tools or test scripts. In these methods, the tester plays the role of an end user and performs the testing to identify defects and unexpected results from the system under test. In manual testing the test cases are generated and testing is done as per the test plan in manual mode. The drawbacks of manual testing are that it is time consuming, less reliable, risky and can provides incomplete test coverage. By using automated testing the testing is done with the help of automated tools and test scripts (test suites). This technique leads to extensive tests of the software quality in completeness. Automated testing is less laborious, much more reliable and faster to execute. There is a test suite development phase followed by a test system execution and subsequent results analysis phases which are partially automated also.

1.2.4 Static and Dynamic Testing

Static testing is also known as specification testing. It is performed while the software is static and not in execution. It is used to test the requirement specifications, design specifications, coding and complete system specifications. The tester can impersonate the customer to check specifications of the system under test. This method is used to check the various attributes such as completeness, correctness, consistency, relevance, feasibility and testability. On the other hand dynamic testing is used to test software once it is in execution. The software is tested for operating as expected and satisfactorily working according to specifications. The actual results from the system under test are compared with the expected results in order to identify and resolve problems in the software.

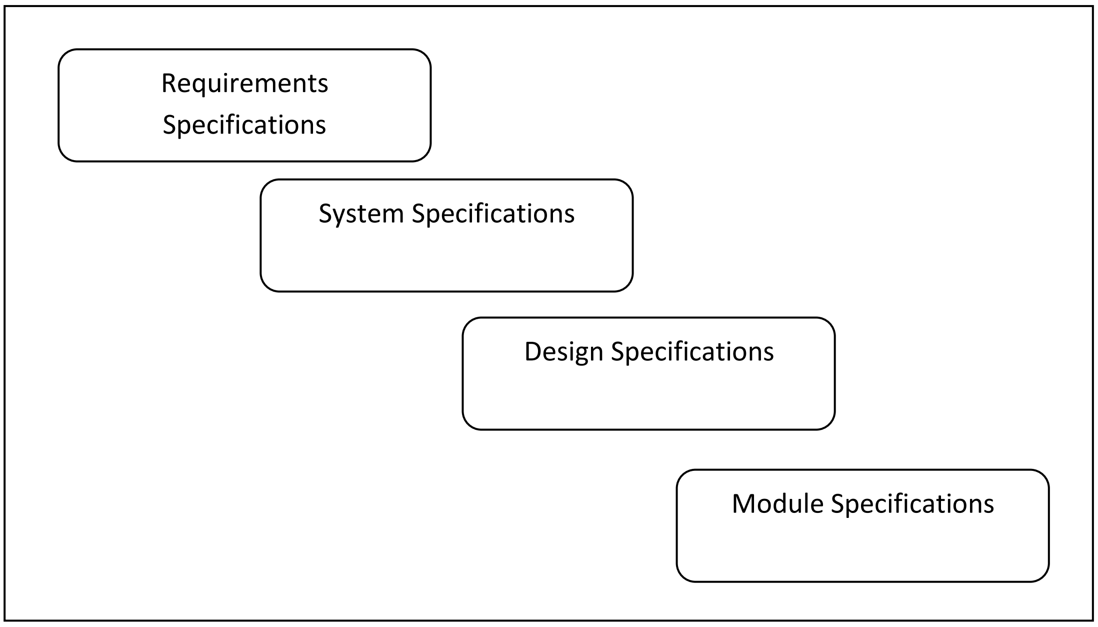


Figure 1.4: Using Specifications in Software Testing

1.3 Web Services

In the last many years there has been a proliferation of web services and applications as part of the Service Oriented Architecture (SOA) based solution offerings prevalent for solving the modern day distributed computing problems. Web Services are considered as a prevalent software integration technology on Internet and Intranet platforms due to its inherent nature of language and platform-independence.

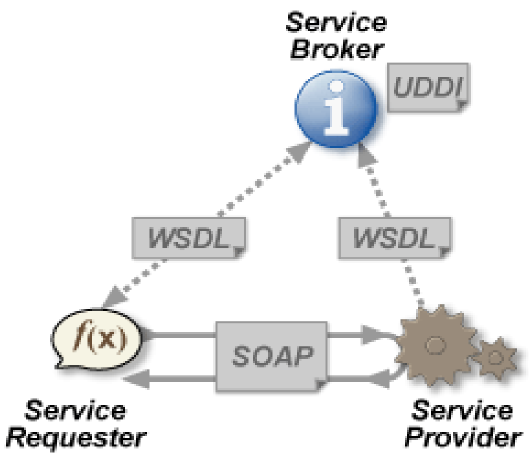


Figure 1.5: SOAP based Web Service [12]

These Software Artifacts have moved applications from a relatively small number of large application deployments to a large number of small application deployments throughout the internet and various enterprise networks. As Web Services adoption keeps rising, more developers are doing more quality testing on them early and often in the development life-cycle to ensure that all the functional modules used by applications work correctly and efficiently all the time.

1.4 Web Services Technologies Area

The language and platform independent nature of Web Services technology brings difficulties in testing their functional as well as non-functional behavior using a uniform set of automated testing tools. Until some time ago the major communication protocol used was Simple Object Access Protocol (SOAP) being mainly XML over HTTP. The data exchanged with web services followed precise formatting rules in the form of XML Document Type Definitions (DTD) or more recently the proposed XML Schema. More recently after its introduction, REST which stands for Representational State Transfer, is an architectural style for networked hypermedia applications, primarily used to build Web Services that are lightweight, maintainable, and scalable. A service based on REST is called a RESTful service. REST has become one of the most important technologies for developing Web Applications. Its importance is likely to continue growing quickly as all technologies move towards an API centric orientation on the web.

Every major development platform now includes frameworks for building SOAP and RESTful Web services. REST however is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol. Recent times have seen the advent of SOAP as well as RESTful Web Service testing technologies which evaluate functionality and load among other aspects to measure how a Web Service performs for single clients and scales as the number of clients accessing it increases. This dissertation applies to automated testing of Web services by usage of the Testing and Test Control Notation (TTCN-3), as well as the popular SoapUI API, software testing technologies and tools. This dissertation work concentrates on the various technology aspects for the development and execution of automation test suites for SOAP and REST based Web Services using these two toolsets.

Table 1.1: REST Service Methods

HTTP Methods	GET	POST	PUT	DELETE
Meaning	GET to retrieve information	POST to add new information	PUT to update information	DELETE to remove an entity
Example	GET /store/customers/1234	POST /store/customers	PUT /store/customers/1234	DELETE /store/customers/1234

1.5 Web Services Testing Area

Web Services testing is different than other application testing methods to some extent. In case reference is taken from the Universal Testing Method as a starting point, in many ways testing a Web Service is no different than testing any other software. Only some of the steps can take on bit of a different flavor in practice as outlined below.

Modeling the Test Space: A clear understanding of the components of the Test System (TS) and the target System Under Test (SUT) is acquired before even an abstract test system is defined. Model-based testing has recently gained attention with the popularization of models (including UML based) in software design and development.

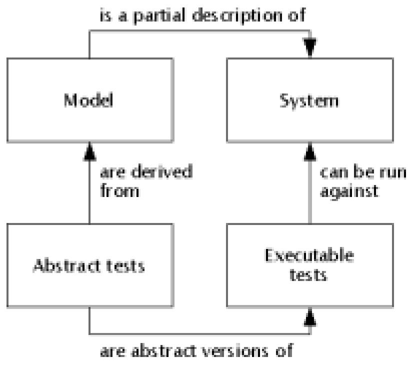


Figure 1.6: Model for Test System [18]

Determining Coverage and Oracles: When testing Web services, coverage becomes important. One cares about all the same coverage cases which are cared about with non-web service testing (scenario coverage, requirements coverage, code coverage, and application data coverage), but now schema coverage gets added to the list also. A schema mapping document tells what data should be stored in what element and when it should be there.

Determining the Test Procedures: An initial problem when performing Web Service testing is management of all the testing and data files (typically Text/XML/JSON formats). A plan is required for handling files that need to be managed manually. Keeping track of Requests, Expected Responses and Actual Responses is important for the Test System. In common cases of having hundreds of test cases, and hundreds of test data files there is a need for programmed scripts to make the management of the test process automated and well managed.

Operating the Test System: There are a lot of great tools available for testing Web Services. SoapUI is a good Open Source option having many technical features for test system execution. As part of this dissertation work much time was spent to speculate using a TTCN-3 based tool for measuring Web Service Quality. There is always the possibility of using homegrown tools written in C/C++ and Java to perform Web Service testing also. Trying more than one tool before settling with a Test System is usually prudent. Once there is adequate and careful comparison between the tools used, SoapUI and TTCN-3, it could provide for an educated recommendation for a Test System Developer.

Evaluating and Reporting the Test Results: Evaluating the results for Web Service tests can sometimes be really easy, and sometimes painful. There are many things required to practice sound Test Reporting such as ensuring at least someone on the test team knows the schema inside out and can see the entire mapping document in their mind while looking at the response files. It’s also found that custom logging (both in Test Execution tool and in the Web Service under test) can help capture the results better. Other concerns that come into play can be authentication, authorization and encryption issues. There is good amount of importance associated with the testing system tools that one uses for testing Web Services, e.g. in this case, SoapUI and TTCN-3 (Titan).

1.6 Web Services Testing Technologies Evaluation

Over the last decade or so there have been continuous improvements in the technology area related to testing of Web Applications in general and Web Services in particular. The primary test tool of consideration for this dissertation, TTCN-3 (Titan), was originally designed as a telecommunications testing platform and used by Ericsson extensively, however it has proven to be an adaptive and powerful platform for Web Application Testing also. On the other hand SoapUI is a popular open source Web Service testing application for Service-Oriented Architectures (SOA) and Representational State Transfers (REST) based systems. Its functionality covers Web Service inspection, invocation, simulation and mocking, functional testing, load and compliance testing.

Major challenges to Web Service testing in the recent past have been integration with existing test frameworks, service-oriented architecture, rich client interfaces, and security vulnerabilities among others. Through careful analysis and practical experience in industrial projects, testing tool developers have developed mechanisms to address each of the above mentioned aspects of Web Application testing. In this dissertation the attempt has been to understand and evaluate TTCN-3 (Titan) and SoapUI toolsets in terms of their facilitation for Web Services Quality assessments along-with the testing models they support. A brief introduction to both the testing tools used in this dissertation is provided below.

1.6.1 TTCN-3

TTCN-3 is a standardized test specification and test implementation language developed based on the experiences from previous TTCN versions used at the European Telecommunication Standardization Institute (ETSI). It's applicable for all kinds of black-box testing for reactive and distributed systems, e.g. Telecom systems (ISDN, ATM); Mobile (Telecom) Systems (GSM, UMTS); Internet (has been applied to IPv6); CORBA based systems. The TTCN-3 technology has built in constructs for, Configuration: dynamic and concurrent test configuration with test components (parallel and serial), Communication: various communication mechanisms (synchronous and asynchronous, message based and procedure based), Control: test case execution and selection mechanisms. The most important concept used is Black-Box Testing with TTCN-3, wherein a Test System (TS) comprising of a collection of Test Cases (TC) and Test Components (TCo) executes against a System Under Test (SUT) or an Implementation Under Test (IUT). The Test System is composed of Test Components (Serial and Parallel ones) along-with an interconnected network of Communication Ports. A particular arrangement of these components in a Test System is referred to as a Test Configuration. The Test Components and Communication Ports support an extensive set of possible Test Configurations suitable for constructing any possible Test System. Each Test Component has its own local verdict (state representing testing progress status), which can be set and read during Test executions. A Test Case finally returns a global verdict indicating the result of a Test System execution. In this particular dissertation work the SUT comprises of components that make up the Web Services domain. The test system interface is towards such a SUT. Using an Automation Tester perspective in this comparison with SoapUI tool, the attempt is to reflect on the Testing of Web Services domain implementing TTCN-3 language constructs listed earlier. However as part of this research work into the usage of TTCN-3 and SoapUI, there are some other aspects from which Titan TTCN-3 is a strong toolset for Web Services Test Automation, which are highlighted in the section below.

Easy Programmability: This standardized testing language (TTCN-3) has the look and feel of a regular programming language but without the complexity that such languages often introduce as it concentrates exclusively on testing of systems. There are many tutorials and courses to learn TTCN-3 for someone familiar with fundamentals of programming. The TTCN-3 standard at ETSI itself provides examples that demonstrate the usage of specific features of the language. The aim of TTCN-3 is to provide a well-defined syntax for the definition of tests independent of any application domain. The abstract nature of a TTCN-3 test system makes it possible to adapt a test system to any test environment. This separation significantly reduces the effort required for maintenance allowing experts to concentrate on what to test and not on how.

Test Data Generation: In this time of fast testing cycles the importance of automated test data generation is paramount. With the power and flexibility of a programming language with a rich data type set, TTCN-3 provides all the facilities for automated test data generation. These test data generation modules go hand in hand with the test case modules which utilize them to perform various test case executions against the system under test.

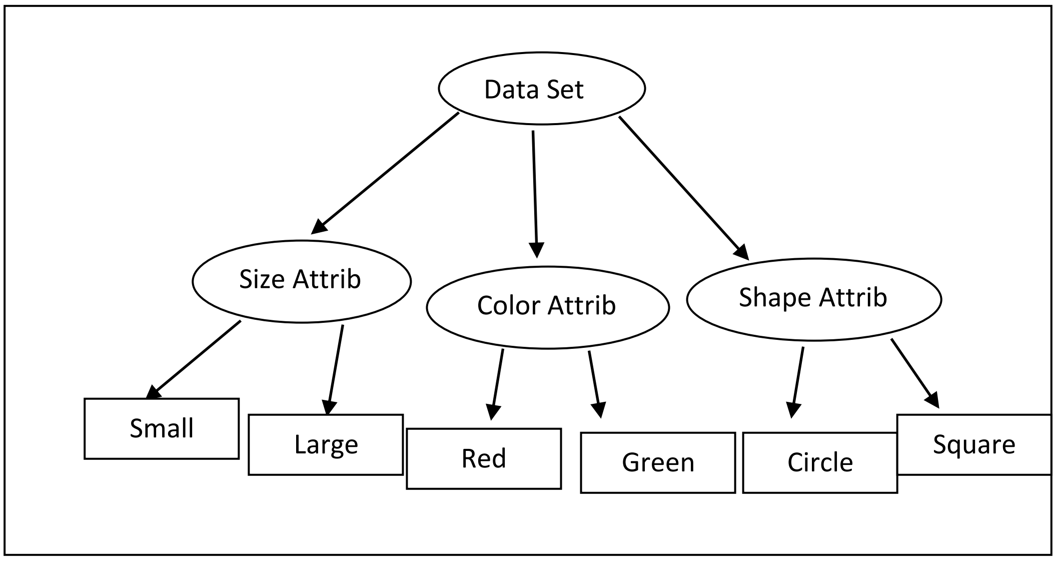


Figure 1.7: Classification Tree Method of Test Data Generation

With a rich collection of types built into the TTCN-3 language standard it is convenient for test developers to write object validation modules, for instance generating the set of possible values to test a web service method with boundary and validation conditions. A rich set of constructs are available for manipulating bits, bytes, numbers, characters, strings and other common objects used as arguments to web methods. TTCN-3 modules can support both application specifications based and program structure based test data generation for use in the test cases. In the case of test data generation based on application specifications methods such as the classification tree method depicted in the figure above could be used to create a minimal set of possible data for use with test cases against a web service system under test. Do refer to the figure above for an outline of this methods algorithmic breakdown structure using test data classification technique.

Numerous methods and algorithms for test data generation have been and could be utilized by the test developer using Titan TTCN-3 based on the knowledge and preference of the tester but as far as language richness of TTCN-3 all could be implemented using the test specifications language constructs.

Test Suite Maintenance: To measure the external attributes of a test system product, execution of the product is usually required, whereas for measuring the internal attributes, static analysis of the test system is sufficient. Since the attempt is to highlight quality characteristics such as maintainability of TTCN-3 test specifications, only internal test system attributes are considered. Internal product metrics can be structured into size and structural metrics. Size metrics typically measure the properties of the number of usages of programming or specification language constructs. Structural metrics analyze the structure of a program or specification. The most popular examples are coupling metrics and complexity metrics based on control flows or call graphs. To assess the overall quality of software, usually averages of metrics are applied. By additionally considering the metrics of individual language elements, it is possible to identify locations of inappropriate usage of programming or specification language constructs. For example, by counting the number of references to each definition, issues like unused definitions can be identified. However, some issues cannot be detected by simple metrics, but need a pattern-based approach. These kinds of issues are called bad smells or code smells. Examples are duplicated code or parameterized definitions, which are always referenced using the same actual parameter values. The results of an issue detection approach that is based on locating patterns of TTCN-3 code smells looks very promising [38]. The focus on quality assessment based on metrics leads to holding the TTCN-3 test suite in good light.

Test Parameterization: TTCN-3 test development and execution system lends by its very architecture to be flexible and adaptable to accommodate many types of testing environments. Module parameters are commonly used to handle parameters that are specific to the System Under for example, the IP address for use with a Domain Name System (DNS) server that is to be tested. Module parameters allow the design of test suites independent of a specific SUT instance. They allow using the same test system executable in different environments, for example, against a local server in a test lab scenario or somewhere out in the field where test execution against the system under test is desired.

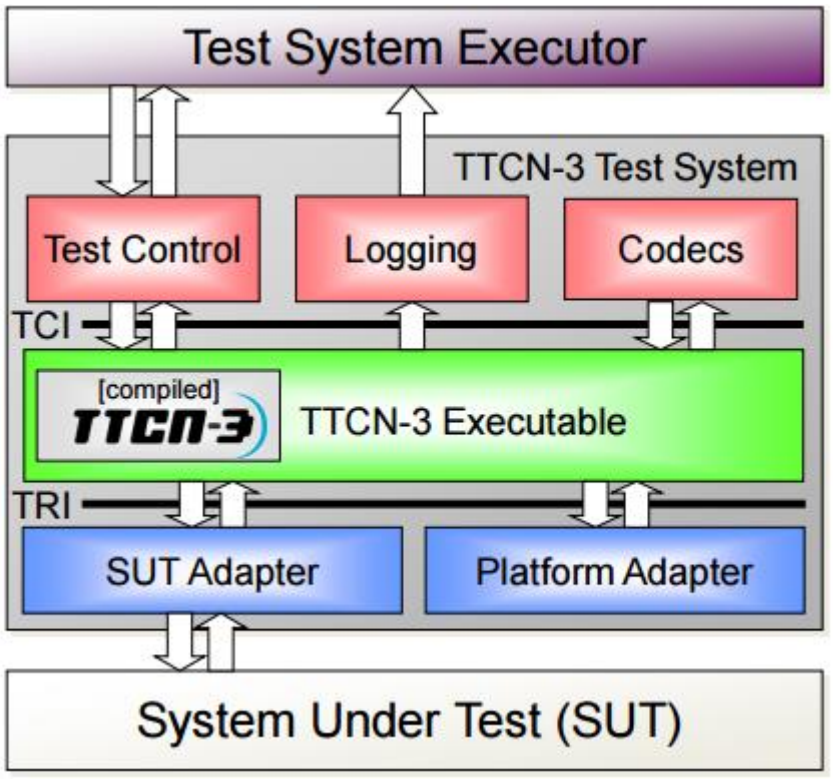
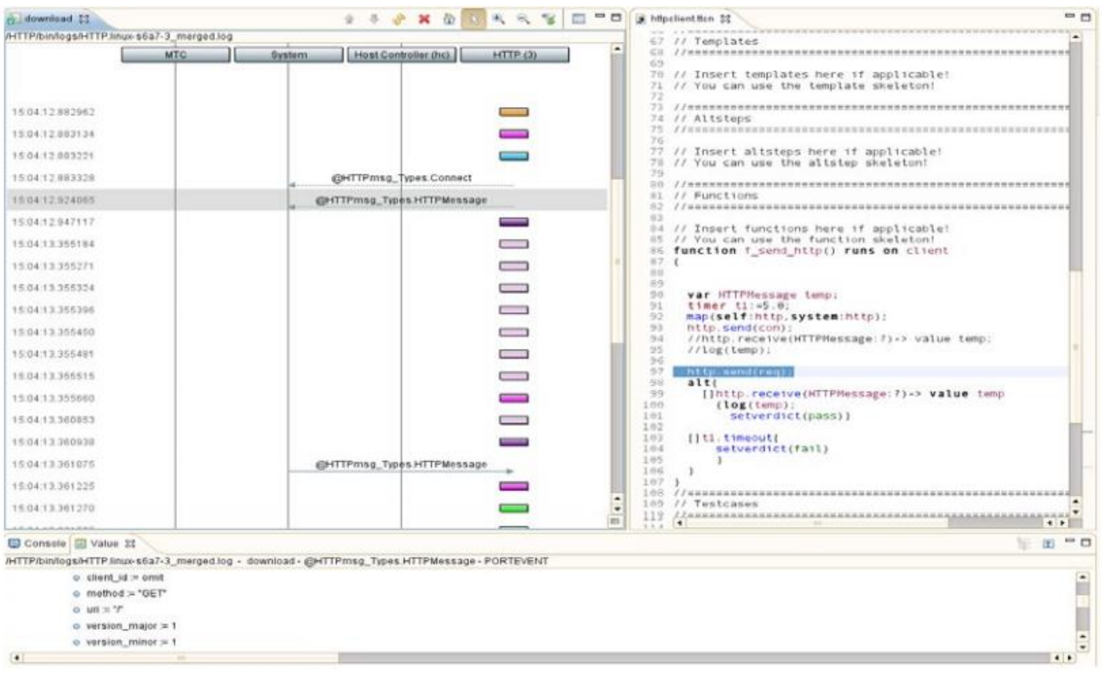


Figure 1.8: TTCN-3 System Adaptability [4, 5]

Test Execution: The TTCN-3 test system is specified in abstract notions and some of the abstract behavior and state constructs need to be implemented. The Titan TTCN-3 tool translates abstract TTCN-3 specifications into executable code that is optimized to run on various processor architectures using C/C++ Compilers and Assemblers. The Titan TTCN-3 toolset also allows to adapt the test runtime environment to test management allowing for pluggable modules such as for test system logging, test execution control that could be customized to suit a test developer and executor. The Titan TTCN-3 toolset also implements the communication and test platform aspects that are required by the executing test system for communication with the system under test and its underlying operating platform as depicted in figure above.

Test Reporting: When using Titan TTCN-3 there are quite a few configurable options for test logging and reporting. It allows for artifacts to be saved to files that could go into a test management repository. Generally, this is a repository communicating and establishing transparency to the test team's activities of the executions during the test cycle, including both defect information and test case run reports information. Titan TTCN-3 allows for graphical message sequence charts as shown in the figure below to capture the execution state completely. Normally, the test development team, test environment support team, and the wider business team find the reports archived in the test management repositories quiet useful. The test reports can be correlated to the test plan as well as the test design which is very useful for quality in large testing environments.



```
/HTTP/bin/logs/HTTP.linux-s6a7-3_merged.log
15:04:12.882962
15:04:12.883124
15:04:12.883221
15:04:12.883328
15:04:12.924065
15:04:12.947117
15:04:13.355184
15:04:13.355271
15:04:13.355324
15:04:13.355396
15:04:13.355450
15:04:13.355481
15:04:13.355515
15:04:13.355660
15:04:13.360853
15:04:13.360938
15:04:13.361075
15:04:13.361225
15:04:13.361270
```

```
// Templates
//=============================================================

// Insert templates here if applicable!
// You can use the template skeleton!

//=============================================================
// Altsteps
//=============================================================

// Insert altsteps here if applicable!
// You can use the altstep skeleton!

//=============================================================
// Functions
//=============================================================

// Insert functions here if applicable!
// You can use the function skeleton!
function f_send_http() runs on client
{


  var HTTPMessage temp;
  timer t1:=5.0;
  map(self:http,system:http);
  http.send(con);
  //http.receive(HTTPMessage:?)-> value temp;
  //log(temp);

  http.send(req);
  alt{
    []http.receive(HTTPMessage:?)-> value temp
      {log(temp);
        setverdict(pass)}

  []t1.timeout{
      setverdict(fail)
      }
  }
}
//=============================================================
// Testcases
//=============================================================
```

```
/HTTP/bin/logs/HTTP.linux-s6a7-3_merged.log - download - @HTTPmsg_Types.HTTPMessage - PORTEVENT
    client_id := omit
    method := "GET"
    uri := "/"
    version_major := 1
    version_minor := 1
```

Figure 1.9: MSC Test Reporting View (Titan TTCN-3)

The TTCN-3 language feature set comprises many more capabilities:

- A well-defined static and operational semantics through its standards.
- A rich type system with many primitive types added to the regular set.
- A powerful built-in data matching mechanism and matching expressions.
- Snapshot semantics that ensure and preserve the order of external event arrival into the test system.
- The ability to define tests with multiple test components executing in parallel.

- Support for message-based as well as procedure-based communication paradigms with synchronous and asynchronous modes.
- Support for dynamic test configurations with test components that can be (re)created and (re)connected on-the-fly even in distributed test systems.
- The ability to specify execution parameters at runtime to ease re-targeting of test suite execution in different testing environments.
- Support for timers which can be customized using TRI interfaces too.
- The ability to automate test execution driven by external sources using the TTCN-3 Test Management interface (TCI-TM).

Flexibility: Prominent flexibility features of the TTCN-3 language are:

- The language is completely independent of technology, operating systems and implementation domains.
- There are no practical limits to the extent that tests or test systems can be adapted to users' needs:
    - Test systems can be integrated easily with the most appropriate test execution management software using TCI Test Management interface (TCI-TM)
    - Test execution traces can be visualized in many suitable formats using the TCI Test Logging interface (TCI-TL)
    - Any encoding scheme can be implemented and integrated using the TCI Codec and value APIs provided by the TTCN-3 standard
    - Test systems can be adapted to any communication mechanism using the TTCN-3 Runtime Interface (TRI) System Adapters
    - Test systems can be adapted to any timing model using the TTCN-3 Runtime Interface (TRI) Platform Adapters
- Highly Scalable Automated Testing System
    - Adaptations can be configured to the current needs while the test scripts remain unchanged and can be used in different development phases

    - Test components can be added to existing test cases to test new interfaces of the SUT as they appear
- Test Components can be used both to test and to emulate system interfaces
- Very Extensible Test Systems
    - Standardized mappings to other external type systems available such as ASN.1 and XML
    - Integration of external functionality is possible using the TRI Platform Adapter
    - Multiple presentation formats are available - textual and graphical

Titan TTCN-3 supports a standalone as well as a distributed model of parallel test components that is quiet missing in tools such as SoapUI. In this mode Test Components of the TTCN-3 test system could execute in a distributed manner over a collection of computers hosting the test system as is depicted in the following diagram.

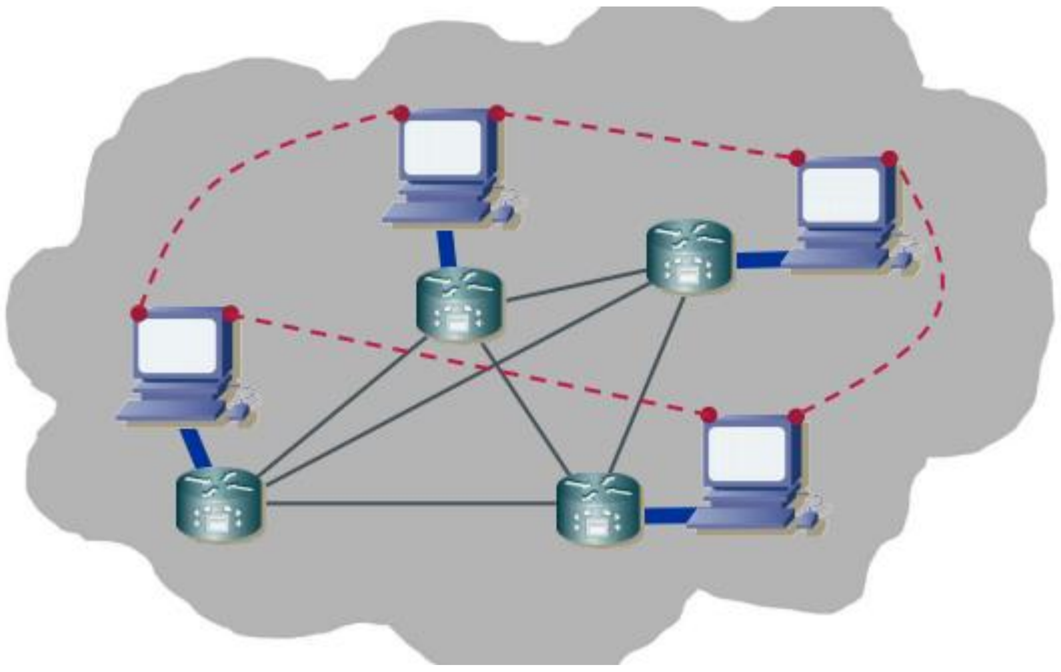


Figure 1.10: Distributed TTCN-3 Test System [30]

There is a central controller associated with the Main Test Component node which communicates and coordinates with all the other Test Components that are running on the remote hosts on the networked system. Such test environments are very useful for designing and executing test systems that are used to run against current day distributed systems under test such as Internet of Things applications.

1.6.2 SoapUI

SoapUI is a leading cross-platform API Testing tool for testing of Web Services [31]. It enables Test System developers to execute automated functional, regression, compliance, and load tests on different kinds of Web based API's. It supports all the standard protocols and technologies to test all kinds of Web based API's including SOAP, REST etc. Its interface is simple and it enables both technical and non-technical users to use it productively and seamlessly. SoapUI is not only a functional API testing tool but also lets one perform non-functional testing such as performance and security tests on different kinds of Web API's as required. It allows test developers to use advanced scripting (custom code depending on the Test Scenario). It has built in functions to perform Security Testing with the capability to perform a complete set of vulnerability scans against Web API's. It pro-actively prevents SQL Injection to secure the databases relied on by Web Applications. It scans for stack overflows, cross site scripting, while performing fuzzy and boundary scans to detect and avoid erratic behavior of the Web Services under test. It has built in support for Load Testing by distributing the Load Tests across multiple agent components. It can simulate high volume and real-world load testing with ease allowing advanced custom reporting to capture various performance parameters for end-to-end System Performance Monitoring. SoapUI has a most comprehensive Protocol Support for Web API's with built in support for, HTTP, WSDL, SOAP, REST, JDBC, and JMS etc. Although most of the functionality is available with the Pro version of the toolset and not the Open Source community version. It is for the popularity reason that it is being used in this dissertation work, as a benchmark for comparison with the TTCN-3 based Test Systems for testing Web Services especially RESTful ones, to determine time and cost benefits of using an open source edition of Titan TTCN-3 over SoapUI.

CHAPTER 2
REVIEW OF LITERATURE

Atkinson Colin, Barth Florian, Brenner Daniel, and Schumacher Marcus [6], present a new approach to software services testing which combines tabular tests specification techniques like Framework for Integrated Test (FIT) with programmatic techniques like xUnit and TTCN-3. This concept is that of ‘Test Sheets’ and two types are described, input test sheets and result test sheets. They contain state conscious inputs to the web service interface and the results generated by the calls to the services respectively. The authors describe a neat method of creating higher level test sheets from lower level sheets using programming concepts of conditions, branching, and parameterization of the sheet elements. In order to apply a test sheet against a SUT, the language independent test definition needs to be transformed into executable code, which can be generated in the form of stubs based on the service description languages such as WSDL. Future work is suggested for extension to test sheets that extends the normal qualitative test results by the measures of quantitative properties of the SUT. This allows to measure timing, deviations and other key characteristics of web services to be measured. Also information could be used to validate the constraint characteristics for commercial third-party web-service providers. It’s suggested that these measures may be combined with a service registry in order to create a QoS-Aware Service Repository.

Huang Teng, Ding Zhizhong, Duan Younan, Chen Yin, and Ding Lu [7], are concerned with the design of a reasonable adapter model for RRC (Radio Resource Control) testing using TTCN-3, which satisfies the requirements of ETSI RRC specifications. After elaborating its mechanism and main functionality in the general structure, an implementation model as running parts of the RRC test case is described. The proposed adapter implements the functionality using reduced resource consumption. By using only a few real machine ports, it has better performance because of system load reduction and has better control of the transmission timing. It enables multi-protocol communication between Test Execution (TE) and System Simulator. Further suggestions are for improving the performance and functionality of the adapter using real time TTCN-3, released as TTCN extensions, to enhance the capability of synchronization of multiple logical channels for the complete LTE RRC Test Suite.

Katara, Mika [8], neatly elaborates the TTCN-3 Control interfaces and the TTCN-3 Run time interfaces and components of the TTCN-3 test system that are controlled and managed by the same, such as Test Management, Encoder/Decoder, Logging System and Platform adapters. The author proposes a neat Communications Management System that bridges the TTCN-3 Test System with a diverse set of Systems Under Test. The architecture of the CMS system is formulated and suggested for implementation and adaptation. Further work suggestions are for improving the performance and functionality of the Connection Management System by providing support for a wide variety of end-to-end protocols in the adaptation layer.

Kumar Dinesh [9], has given a conceptual insight on operator action validation using a validation slice created independent of the real system interfaces, i.e. Web Services. The V Model of test driven parallel development is elaborated in the context of Web Services testing. Web Applications testing approached from Functional, Browser Compatibility, Performance, Transactions, Compatibility & Security aspects is suggestively reviewed. TTCN-3 is compared as a Web Testing tool against PureTest, highlighting the use of recent TTCN-3 HTML related extensions, which lead to more extensive and accurate Web Testing using TTCN-3. However PureTest is upheld as a better Web Testing tool as of writing of the paper. Future Work implicitly recommended is that developments and research is welcome in the area of fast HTML (web protocols) parsing and robustness in the Web Testing components of TTCN-3.

Liu Yang and Hang Bei [10], highlight that dependencies among the abstract test case/suite, the codec entities and adapters (SA and PA) are rigidly configured in Test Systems. There is no explicit semantic definition of these dependencies in the TTCN-3 standards so far. The Solution suggested: Using a Capability Description Language for specifying the dependencies among the abstract test case/suite, Codec entities, Adapters (SA and PA), and a Test Adapter Framework, to automatically map, select, and load the Adapter which complies with the requirements of test cases. This is based on the Capability Description Language and is transparent to the test system. The Test System Interface (TSI), Codec (CD), System Adapter (SA) and Platform

Adapter (PA) entities have associated capability description language files describing the communication, data and external functionality attributes of the same. Using these as part of configuration management, the selector component queries the TE for attributes, based upon which it acquires references to objects through a mapper, which is brought in by the loader from CD, TA, or SA jars. Further work is suggested on the lines of better implementations for CDL based Test development and execution tools.

Rentea Cosmin, Schieferdecker Ina, and Cristea Valentin [11], state the utility of TTCN-3 as representing a Test Design with its Abstract Test System (ATS) and leaving the Test Implementation specifics with the Codecs & System Adaptation modules. Testing user experience with TTCN-3 is considered at the end-user level with server content that the general HTTP protocol supports. The concept of Mapping Repository for storing Web Target Test Data is specified to automate the process of web data extraction. The other significant concept is that of the TWeb module of TTCN-3 which is researched in relation to the Mapping Repository and the TWebCodec as well as TWebAdapter components. One of the next things suggested to be done is the actual integration of WebTestGUI (Mapping Repository) with extensions suggested, and TWeb (Web Testing Implementation components) as Eclipse plug-ins. Suggestion is that distributed test setups for efficient load, performance, scalability tests are possible using the TTCN-3 test platform. Also similar implementations with SOAP based and other Web Services is suggested.

Schieferdecker Ina and Stepien Bernard [12], consider both functionality and load aspects for testing how a Web Service performs for single clients and scales as the number of clients accessing it increases. This paper discusses the Automated Testing of Web Services by use of the Testing and Test Control Notation (TTCN-3). A mapping between XML data descriptions to TTCN-3 data is presented to enable the automated derivation of test data. This is the basis for functional and load tests of XML interfaces in TTCN-3. The mapping rules and prototypical tools for the development and execution of TTCN-3 tests for XML/SOAP based Web Services are elaborated. A further key element of the test framework is the automated translation of XML data to TTCN-3, so that test skeletons can be generated directly from the specification of the Web service. The conversion tool together with the TTCN-3 compiler and execution environment TTthree provides us a complete tool chain for test data type generation, test development, implementation and execution. The test framework has already been used successfully for selected Web services. Future work involves furthering elaborate methods for test data generation. In particular, the classification tree method could be investigated for potential extension towards the generation of templates for SOAP responses. In addition, the test framework can be enhanced to deal with further elements of Web Services like the specifics of WSDL and UDDI.

Schieferdecker Ina and Vassiliou-Gioles Theofanis [13], layout the transition from closed box testing systems to open ones driven by nature of current systems to be tested. Clearly stated is openness of specification in terms of Data Types, Configuration, Behavior, and Control in a Test System.

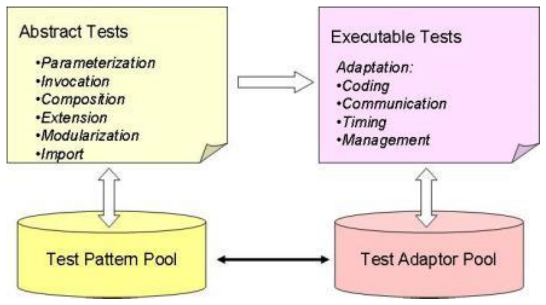


Figure 2.1: Abstract Test and Execution Systems [13]

Introduced to the TTCN-3 reader is the concept of a test pattern which is used as a building block for extensions and frameworks which are reusable with appropriate parametrization. The GPRS target device, G20, related test scenarios, explain adding a new device to the Test System and also adapting existing TTCN modules to TTCN-3 with tool support (in this case TTwb, TTspec). Interesting reference is the use of CORBA based API to run the test system against the G20 device. A Case Study Description based (Test Framework) is suggested and further work is advised along the lines of developing, extending, or using TTCN-3 test frameworks that are accepted, and which tool vendors can integrate for required built-in support.

Schieferdecker Ina, Din George, and Apostolidis Dimitrios [14], elaborately present a flexible Test Framework for Web Service verification realized using TTCN-3. The techniques and methods used for this test framework elaborate on usage of a TTCN-3 compiler, an XML-TTCN-3 conversion tool, and a Test Adapter for XML/SOAP interfaces to Web Services. The adapter explained is generic and thus enables testing of any Web Service using XML/SOAP interfaces. The paper concentrates on functional and load tests against multiple interfaces of a Web Service, giving an elaborate example of a distributed load test with distributed test components.

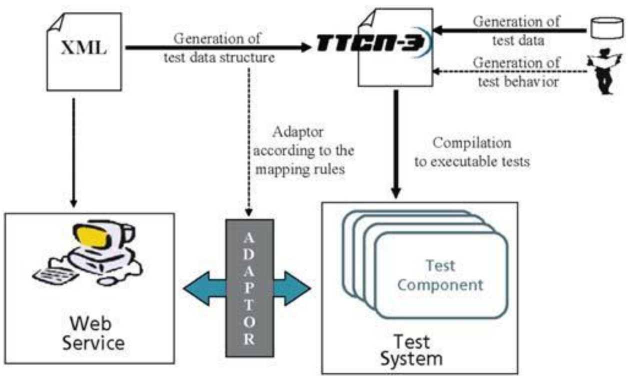

XML
Generation of test data structure
TTCN-3
Generation of test data
Generation of test behavior
Adaptor according to the mapping rules
Compilation to executable tests
Web Service
ADAPTOR
Test Component
Test System

Figure 2.2: TTCN-3 Test System Architecture [14]

Future work suggestions: Test Patterns beyond the presented functional, service interaction, and load tests are suggested to be investigated using a similar framework. Further collaborative and exploratory methods for test-data and test-behavior generation, in particular, the classification tree method is suggested to be investigated for potential extension to the generation of TTCN-3 templates. Also automated test generation from UML Test Models is suggested.

Schulz Stephan [15], provides very useful insights into the area of TTCN-3 library development with the objective of code reuse and improving the quality of testing code artefacts. The author introduces a methodology for TTCN-3 library development based on a layered approach, wherein closest to a TTCN-3 test suite is the TTCN-3 Test Suite library, followed by TTCN-3 Interface Libraries & other Common TTCN-3 Libraries forming layers respectively. The insightful examples given on client & server component synchronization methods, and reuse of test behaviour inspire a test developer to use these methods for implementing quality TTCN-3 library modules. Further work is suggested along the lines of exploring TTCN-3 language construct enhancements, such as intermediate verdicts for test components, that would make reuse of TTCN-3 state and behaviour more feasible for TTCN-3 developers.

Stepien Bernard and Peyton Liam [16], give an insightful introduction to the application of TTCN-3 in the area of Web Application Testing, as per functional, quality of service, and performance testing's. Clarity is provided on the mechanisms and utility of Test System Abstraction for Web Testing using tools such as HTMLUnit and JUnit for adapted implementations with a Web based SUT. Secondly, the practical mechanisms of testing a Service Oriented Architecture (SOA) application along with its component services is exhibited using TTCN-3. Thirdly, Web Security testing methods are explored using TTCN-3 examples of penetration testing for Web Sites, emulating both genuine user behaviours and attacker behaviours in TTCN-3 test components. Further work is recommended along two lines: Implementation work, one area is of exploring use of test agents' architecture to address quality of service issues related to response times. The root cause could potentially be with any of the components in a SOA Web Application. Narrowing down this problem using a TTCN-3 test suite is recommended. Architectural work (Standards Change also), Recommendation is for TTCN-3 standard to adapt to needs of Distributed Systems Under Test (in context of multiple Web Sites or Web Applications being tested by a single test-suite concurrently).

Stepien Bernard, Peyton Liam, and Xiong Pulei [17], introduce the reader gently to their TTCN-3 Web Testing architecture as comprising of, a) Abstract Test Suite: with functional, integration testing logic, b) Implementation: with Codecs & Adapters, which utilize frameworks such as HTMLUnit, HTTPUnit, JUnit, ServletUnit, etc., to map the abstract test system with an actual web application System Under Test (SUT). They present the concept of parallel coding and testing process, in which both paths are progressing on the same functional specifications for the web application. The abstract test suite (ATS) acts independently from the coding/decoding, communication and presentation details using templates and pattern-matching to capture relevant presentation and communication details. TTCN-3's data types and set-based operations are demonstrated to serve as powerful constructs for tracking and verifying the information management done by a web application independent of implementation details. Further work is recommended towards the extra level of sophistication and skill required from test developers who wish to use this approach (with specific web applications and test frameworks), with the benefits gained in productivity (reuse) and effectiveness making this approach worthwhile for other test development.

Stepien Bernard, Peyton Liam, and Xiong Pulei [18], evaluate the usage-suitability of TTCN-3 as a modelling language for Web Penetration Testing. It's demonstrated that the inherent abstraction features in TTCN-3 make the process of generating Web Penetration test campaigns simpler. Especially, the ability of combining separate models for relevant web vulnerabilities and web application functions into a generic web abstraction model and a TTCN-3 test framework is explored. The process of generating web penetration tests is differentiated from functional testing by basing the web penetration test model on the layout-foundations of the functional test model once it's formulated and specified. The web penetration test cases are derived from a combination of the observed behaviour of a fully functioning web application and a model of web vulnerabilities. Two models are built around TTCN-3 in order to enable implementation of the model test cases. The TTCN-3 language as a testing language has its own model that is based on the concepts of separation of concerns between an abstract and concrete layer and within the abstract layer there is also a separation of concern between behaviour and conditions governing behaviour revolving around the TTCN-3 concept of a template. Also a specific abstraction model is constructed using the TTCN-3 language to describe web tests using abstract data types separating application data at an abstract level from HTML implementation and encoding details. Further work, addresses two types of penetration test cases: SQL Injection and XSS attacks. The author's suggestion is to use the shown concepts and techniques for other types of penetration tests, as well as addressing other classes of security attacks and vulnerabilities.

Sun M, Zou J, and Ma Shi H [19], stress that complexity of modern telecommunication systems leads to the need for thorough and systematic software testing. Software testing is an expensive and time-consuming task involving specification of what and how to test, and preparation of test descriptions in a format that is accepted by the test equipment. In order to cut down the cost of manual testing and to increase its reliability, steps are taken to automate the whole test case preparation process. In this paper, they describe a model for automatic test generation, and description of an implemented solution for automatic generation of TTCN-3 test scripts based on parsing SIP call traces. The implemented solution is tested using TITAN, a TTCN-3 test execution environment developed by Ericsson. Future work is suggested to implement codecs with support for more protocols such as the data transmission ones used around the SIP protocol.

Werner Edith, Grabowski Jens, Troschütz Stefan, and Zeiss Benjamin [20], emphasize increased use of Web Services for critical applications introducing a need for efficient testing approaches to assure their quality. The TTCN-3 testing language is emphasized as well suited for black-box testing of such distributed systems. Also due to its abstract test specification methodology, it allows easy adaptation to different Web Service frameworks or platforms (e.g. Java, .NET). This paper primarily presents a mapping mechanism from the Web Service Description Language (WSDL) to TTCN-3 and guidelines for a corresponding automated translator. By using the WSDL description of a Web service, it is possible to derive an Abstract Test Suite (ATS) in TTCN-3 which provides the necessary abstraction from the implementation, platform and the communication details. Author’s goal is to make Web Service testing in TTCN-3 become more reliable, comprehensible, consistent and efficient. Automated translation modules would enable the test developer to concentrate on actual Web Service Test Development and avoid possible human errors. Future work can be an attempt to couple the existing test case composer (WSDL to TTCN-3) with a process description like the Business Process Execution Language (BPEL) to automatically generate more advanced test cases.

Yanwu Tang [21], has focused on using TTCN-3 Codec interface implementation to convert abstract data format (TTCN-3 Test System) to SUT (System Under Test) specific binary data formats. Codec development significance is stressed as a major part of all real test system development. Research focus is on improving the efficiency of Codec development for test and tool developers. The VCL system tries to overcome the limitations of the TTCN-3 Control Interface by providing a richer Intermediate Data Type Representation building over the TTCN-3 standard primitives. Future work is suggested to implement compiling, executing, controlling, logging functions of TTCN-3 test system as Web Services. Also suggested is work for easy-to-use system adapter with high configurability to make writing system adapters much easier in Loong Testing (a TTCN-3 tool set from USTC).

Zhao Huiqun, Sun Jing, and Liu Xiaodong [22], guide on usage of Business Process Execution Language (BPEL) for composing the structure of Web Services and their work-flow. BPEL is introduced as a standard for Web Service Architecture with particular attention to utilizing the same for Quality Control of these services. Another tool introduced is, Labelled Transition System (LTS), for representing the BPEL model. The model checking for logical correctness is done with the help of a commercial toolbox CADP which is described in brief. A facility is introduced to generated TTCN-3 Behaviour Trees (BT: using TTCN-3 ALT constructs) from the BPEL-LTS systems. So logical model testing can be done prior to using TTCN-3 to test the web-service systems represented by BPEL-LTS constructs. Authors attempt to list step by step, the equivalence between LTS and BT for generating test case from model checking counter-examples. For further work, the suggested direction is to investigate and develop an approach to improving the reliability during web service evolution cycle, finding the possible methods of how to generate traces based on the clues where cooperation protocols are changed.

Malik Vinita, Gahlan Mamta [23], familiarize some of the popular Web Testing tools such as Quick Test Professional (QTP), Selenium, Test Complete, SoapUI etc. Providing clear advantages and disadvantages of using these in the context of Web Application and Services testing. The authors compare the Test Systems along with their host toolsets based on practical parameters of measurement such as capability of tests scripts generation, script reusability, cost, execution, test result report, easy learnability, and execution speed. The architecture of the Test Systems is suggested to the reader for implementation and adaptation. Further work suggestions are for performance and functionality based tool comparisons to provide support for a wider variety of Web Testing scenarios.

Kumar Ravi, Singh AJ [24] describe the features of popular Web Application Testing tools such as Apache J Meter, SoapUI Pro, WCF Storm, Soap Sonar etc. for functional and load testing, listing some advantages of using each in the context of Web Services testing. The authors compare the Test Systems based on these tools on practical parameters of measurement such as response time, throughput, and average response time. It is concluded that for the test web service the test tools SoapUI Pro and

Apache J Meter perform best with SOAPSonar following close by. Further work suggestions are for performance and functionality based tool comparisons to provide support for a wider variety of Web Testing tools and scenarios.

Wala Tanuj, Sharma Kumar Aman [25], describe features of Web Application Testing tools such as Apache J Meter, SoapUI Pro, WCF Storm, Wizdl, Web Inject etc., listing features of using these in the context of Web Services testing. The authors have compared the Test Systems based on practical parameters of measurement such as response time, throughput, average response time and input-output validity. It is concluded that for the test web service the test tool Apache Jmeter performs best with SoapUI Pro following close by. Further work suggestions are for performance and functionality based tool comparisons to provide support for a wider variety of Web Testing tools and scenarios.

CHAPTER 3
SCOPE OF THE STUDY

The work described in this dissertation is performed to explore and evaluate the possibilities of utilizing alternative testing technologies such as TTCN-3, along-with analysis of alternate techniques for Web Service Testing such as of functional, and performance nature. For over a decade, qualitative and quantitative Web Testing measurements have been performed with a variety of tools in the industry. In recent years, TTCN-3 technology, which was a golden standard for telecommunication standards testing at European Telecommunication Standards Institute (ETSI) has also been gaining popularity for the quantitative measurement of Web Applications and Services. One of the reasons for this trend is the possibility to use modern TTCN-3 technology with add-on implementations for testing Web based interfaces which improves the precision and accuracy of the testing results. The well-established techniques and principles used for functional and performance testing, are already commonly used with SoapUI in the area of Web Services testing where it is firmly established as a very popular tool. However, the analysis in this dissertation is of a qualitative or semi-quantitative nature to allow successful adaptation of TTCN-3 technology for high quality standards of regulated Web Services testing, exploring the needs for further innovation of approaches used in this area. In this dissertation work, the main goal is to address several Web Service Testing challenges commonly faced with functional and performance aspects of the testing process in order to help extend the possibilities of utilizing TTCN-3 in this field as a real alternative for tools such as SoapUI. Since SoapUI is currently much more popular than TTCN-3 for Web Services Testing, the possibilities to improve the testing area with the use of alternate methods is investigated and proposed with functional and performance viewpoints as an aide to practicing Test Suite Developers. Since TTCN-3 is a multi-platform technology addressing a wide range of black-box testing systems its multi-dimensional usage is intended to be highlighted.

CHAPTER 4
PRESENT WORK

Test automation is an attempt to automate the development and execution of test artifacts for measuring the quality of a system under test. It is used to verify that the functions of a system under test are functioning correctly in normal circumstances and behaving soberly in abnormal conditions. Test automation is a very essential part of modern day testing life cycles and is becoming a regular part of the product development cycle. As the software development market becomes more complex and competitive along with more demanding consumers, the time to market for products is becoming shorter. Software testing can be very time consuming and can take up lot of cost in a project. In order to make software testing more effective and efficient there is an ever-growing need for automation in testing processes. Implementing automated testing with effective test case development and execution against the system under test and subsequent automated analysis of results provides a higher rate of fault detection, thereby leading to higher software quality. With automation methods test development can resume earlier in the product development cycle and improve the chances of faults being detected in the early phases of the development. This leads to higher quality of software products and higher levels of satisfaction among the customers. In order to provide automated test development and execution of Web Service Testing Suites tools such as SoapUI and Titan TTCN-3 are used. SoapUI is a well-established player in this area, but our attempt is to bring forth the benefits of using TTCN-3 (Titan) in the area of Web Services testing.

4.1 Problem Formulation

In the recent past of software development much research has been done on improving the software engineering process. Automated Software Testing is one of the important areas of research in the software engineering space. It is required by the software industry as it moves towards agile software development methods. Large and complex software projects involve a large inventory of testing artifacts, their development, their test data, their execution results and reports which need to be maintained throughout the software life cycle. Such test suites provide functional, regression, performance, load, stress and other types of testing against the system being developed. Hence there is a need for increased test automation across organizations, industries, and software technologies. Following this thought process it was found to be interesting to follow test automation in the area of Web Services. Web Services are fundamental to many of today's software solutions and much research has been done by various researchers on automated testing of Web Services. A common problem encountered is the close association of the test cases with the system under test. This lead to exploring technologies like TTCN-3 which provide the capability to write abstract test suites making the test artifacts reusable over different TTCN-3 implementations. To overcome the problem of tight binding between test cases and systems under test, I explore the testing of RESTful Web Services using the TTCN-3 implementation from Ericsson. I used a leading open source toolset SoapUI as performance benchmark for measuring effectiveness of using TTCN-3.

4.2 Objectives of the Study

The objectives of this dissertation work arose from discussions, considerable literature survey along with exploration of the contemporary technologies in the Web Services testing area. The dissertation objectives are primarily focused on:

1. To apply TTCN-3 technology to testing of Web Services, and compare to determine the effectiveness of using it over existing techniques prevalent with SoapUI.

2. To determine the effectiveness of using TTCN-3 over SoapUI for reducing Test Suite development cycle in terms of time spent and cost incurred by Test Developers.

4.3 Research Methodology

Test Automation using the TTCN-3 development and execution techniques is used to make the Web Services testing process efficient. The main aim has been to present methods of developing an extensive and rich test suite to enable the tester to find out maximum number of faults in the least amount of time i.e. as early in the software development life cycle as possible. At the same time the testing budget has to be considered also as organizations attempt to strike a balance between the extent of testing on products and the quality of those products. Hence the attempt to illustrate a method of testing Web Services using TTCN-3. This leads to using newer techniques which are more efficient as compared to the other alternatives (such as SoapUI). The methodology that is followed for this comparison of Web Service testing techniques is concisely elaborated in the following steps:

Developing the Research Problem: As stated earlier the objective is to compare and determine the effectiveness of using TTCN-3 over SoapUI for functional and performance testing of Web Services. The clear intention is to explore gains in efficiency with test development cycles, test coverage, and simplification of the test framework, leading towards a more effective Web Services testing process. It is this dissertation's direction to use open source tools, TTCN-3 (Titan) [30] and SoapUI [31], in a test development environment setup consisting of a uniform platform on a personal PC. The system used was a Windows 7 64bit pc system with 4GB of RAM as a platform to run both toolsets, Titan TTCN-3 and SoapUI. While performing literature survey related to software testing in general and web services testing in particular, it was realized that automated testing tools satisfy the test requirements of current software. In an automated test environment there is an ever-growing need to create, organize, and manage a collection of test artefacts against a particular system under test. Following this line of thought I explored the possibility of using open source tools for fulfilling this need. During this attempt at exploring I came across the use of TTCN-3 in a diverse set of testing areas and its consequent capability to adapt to various testing requirements. There was interest to explore the Web Services testing area, and found much literature related to testing of SOAP based web services with TTCN-3. Interest evolved in exploring the application of TTCN-3 to the testing of RESTful web services. At this point it was decided to explore TTCN-3 for RESTful services testing and especially to use SoapUI as a benchmark to compare. After review of research papers on testing of SOAP based Web Services using TTCN-3 and the techniques used by TTCN-3 to separate the test case design and coding from the implementation and bindings with the system under test. It came to light that such a technology could be applied quiet efficiently to automated testing of RESTful Web Services. Upon review of certain recent research papers on comparison of Web Service testing tools it came to light that SoapUI was a prominent player in this area and could be a great scale to compare TTCN-3 performance against. There came a belief that the testing technique with TTCN-3 would provide a better result than the SoapUI toolset.

The Basic Research Execution Plan: The approach followed is to develop common Systems Under Test. Started by using certain public Systems Under Test (target set of Web Services) on cloud based platforms to emulate Web Services on Web Servers in a real world environment. In order to perform equivalent tests on the system under test from two separate test environments such as those of TTCN-3 and SoapUI, both the toolsets were installed over the common platform which was Windows 7. The Test Systems (both for TTCN-3 and SoapUI) were setup on the local student PC with TCP/IP access to a Tomcat Web Server. The test scenario case studies were designed and developed first, followed by the coding of test suites using both the client tools being used for comparison, TTCN-3 and SoapUI. The developed testing artefacts would be executed against the target SUT (the common set of web services) to generate and record the required test reports for detailed analysis and interpretation of test execution. It was my intention to focus on the functional and performance aspects of both the toolsets, along-with their scalability, simplicity, extendibility and flexibility for a test developer and executor. A User Management RESTful web service is used as a model system under test, with a minimal database to maintain a set of users. I have developed and executed common test cases in both the environments to determine the development and execution advantages that come to light with the use of TTCN-3 over SoapUI. There was an attempt to differentiate development methods used with both tools, focus on the performance gains with a C++ implementation of TTCN-3 as with the Titan toolset. Other aspects such as scalability, extendibility, and flexibility in implementing automated test systems is also considered during comparison of the TTCN-3 and SoapUI toolset.

Collecting Relevant Test Reports: After the development of Web Service system under test case studies and subsequent coding of test cases based on them, and upon their subsequent execution in the respective tool environments there was an earnest attempt to capture execution parameters in the test reports obtained, relevant to test coverage, performance parameters etc.. I primarily obtained individual test case and overall test suite execution times in terms of milliseconds spent during execution of similar test systems on both toolsets. Also prior to this there was some theoretical measure of development time devoted to development of the Test Suites using each tool prior to setup of the execution environment using measures such as Lines of Code (LOC) and man days of development time, and count of functional modules developed. The primary test reports are in the form of log records from the execution of the test cases in both the toolsets against the common system under test.

Analysis of Comparative Test Results: In this stage the activity was to perform a comparative evaluation of development cycle results and execution cycle results for both the client tools. The development cycle comparisons would be based on theoretical models of man day efforts and lines of code among other parameters, while the execution parameters would be compared on more granular terms based on machine performance cycles and resources consumed by test execution. I attempt to highlight the strengths of the TTCN-3 toolset and present its utility in terms of a test automation toolset on grounds of better execution time and richer development environment for test systems.

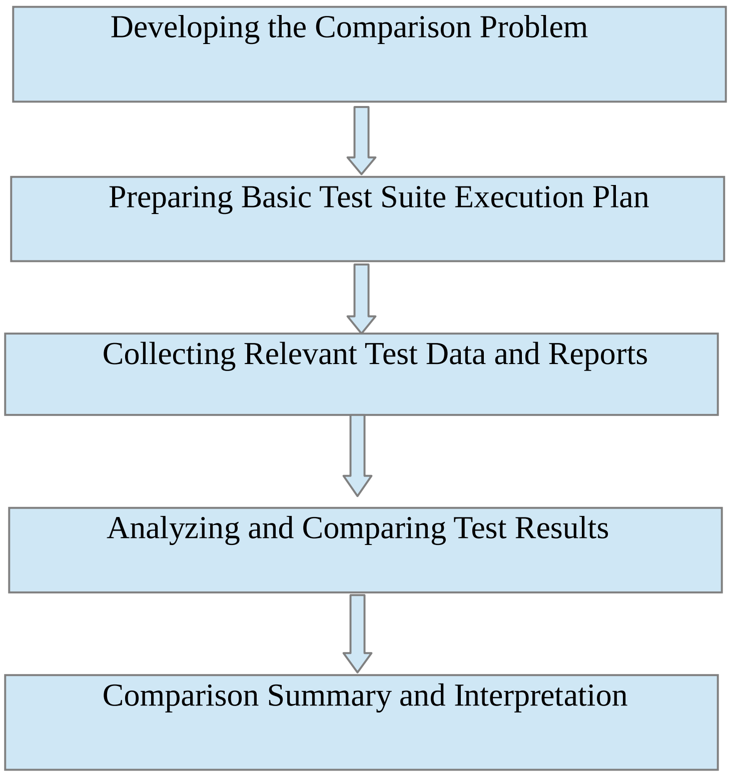

Developing the Comparison Problem
Preparing Basic Test Suite Execution Plan
Collecting Relevant Test Data and Reports
Analyzing and Comparing Test Results
Comparison Summary and Interpretation

Figure 4.1: Flow of Methodology

Comparison Summary and Interpretation: In this stage there was an attempt to be impartial in efforts to get and report accurate measures from the previous steps. But since TTCN-3 toolset from Titan is expected to give better runtime results it puts it in good light against SoapUI as an alternative test system development environment. Reporting on these matters is to be followed by usage of simple comparison models for comparing development cycle time and costs for the developed test suites using both the client tools. Following this are present measures of the effectiveness of TTCN-3 over SoapUI in terms of the development and execution results for functional and performance tests on the target Web Services.

At this point in this dissertation work using the outlined research plan and methods above, the theoretical basis of Test Systems to be developed relies on Web Services related research papers especially by Ina Schieferdecker and Bernard Stepien and other literature on TTCN-3 supporting their advice. Some of the testing methods are 'tried and tested' but some of the areas in this dissertation involved much experimental work, to see if some anticipated assumptions work in real circumstances such as the application of TTCN-3 to testing of RESTful Web Services. But a careful recording of results and subsequent analysis is aimed for reliability on quality of the dissertation results. The primary research methods used are of test system design and development, followed by execution of such a system against a target System Under Test (SUT). The key challenge that was faced related to capturing and/or obtaining reliable measurements of development and execution parameters. I mostly relied on the execution time result logs for performance timings of test suites.

There is a need to find out more on the characteristics and behaviour of Web Service Testing Systems through the application of more data type rich and abstract test development tools such as TTCN-3, which are platform independent, having a rich set of constructs most suitable for efficient testing of multiple target systems including Web Services. There is a growing trend for seeking reliable standards based test systems that exhibit abilities for testing in multiple system domains. Most testing technology systems in use today are based on a tight binding between the Test System and the Systems Under Test including popularly available Web Service testing tools such as SoapUI, which is compared to the more recently emerging field of Testing and Test Control Notation (TTCN-3) based test systems. The expected outcome is to quiet gladly highlight on the utility and usefulness of using TTCN-3 for testing of Web Services especially in an Enterprise Applications scenario. It would enable an organization to benefit from a standards based system which could be applied to multiple test domains thereby saving test developer efforts as well as providing cost savings to the IT departments of these organizations.

CHAPTER 5
RESULTS AND DISCUSSION

In the previous studies it has been reported that, different tools and techniques are utilized to provide reliable Web Service testing and hence quality of functionality and performance. In light of shorter test development cycles due to time to market limitations it becomes very important to utilize tools that find faults in the shortest period of time. The higher the number of faults detected in unit time the greater the quality of the product to market. It is very useful for a test developer and executor to be able to use a rich toolset for performing the required Web Services testing.

TTCN-3 was initially designed as a telecommunications testing language but several aspects of its framework are very relevant to Web Application testing, and provide significant advantages over more traditional approaches to Web Application testing today. The most significant aspect of TTCN-3 is its architecture in which there is a separation of concerns between the abstract TTCN-3 specification languages where test logic is specified (based on test design) and the implementation layer (test runtime) where adapters and codecs formulate test messages and parse responses to and from a System Under Test (SUT).

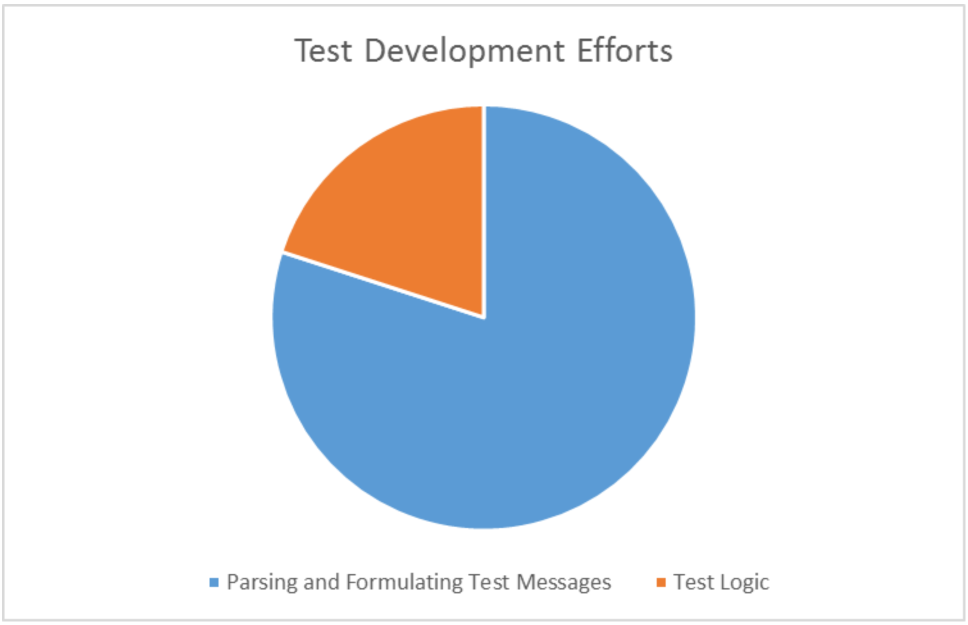
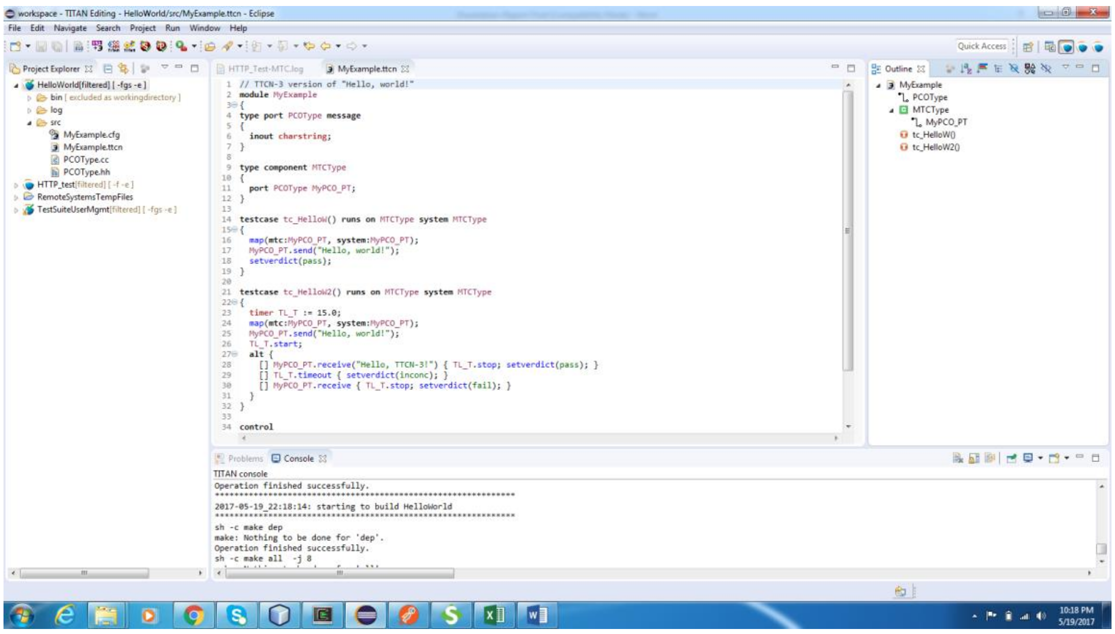


Figure 5.1: Distribution of Test Development Efforts

This allows TTCN-3 to be flexibly integrated with specialized unit testing frameworks and also addresses many of the challenges associated with sophisticated Web Applications testing. This architecture provides significant reuse and dramatically reduces the size and complexity of test logic. Typical Web Service testing applications spend roughly 80% of the code on parsing and formulating of test messages and responses. In TTCN-3 much of the constructs (adapters and codecs) are reusable thus reducing this coding effort, while separating out the 20% of the code that is test logic making tests easier to maintain and understand (as shown in the above pie chart). In this dissertation work the attempt is to display this effectiveness of TTCN-3 by comparison with a well-established open source testing tool, SoapUI, used currently in the industry for Web Service-Oriented testing.

The aim of this dissertation work is to demonstrate the objectives listed earlier by applying a Web Service development model based on TTCN-3. This model of Web Service (especially RESTful) testing environment would help practicing test developers in selection of TTCN-3, based upon its merits and functional value for test planning, design, development, and execution and reporting in their projects.



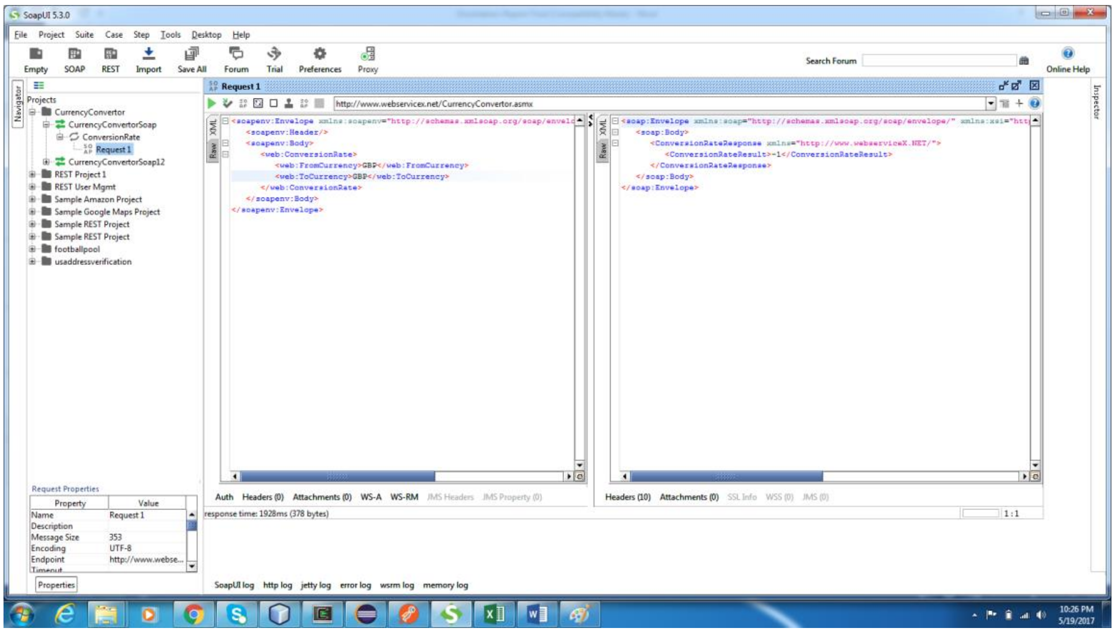
workspace - TITAN Editing - HelloWorld/src/MyExample.ttcn - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
HelloWorld[filtered] [-fgs -e]
bin [excluded as workingdirectory]
log
src
MyExample.cfg
MyExample.ttcn
PCOType.cc
PCOType.hh
HTTP_test[filtered] [-f -e]
RemoteSystemsTempFiles
TestSuiteUserMgmt[filtered] [-fgs -e]
HTTP_Test-MTC.log
MyExample.ttcn
Outline
MyExample
PCOType
MTCType
MyPCO_PT
tc_HelloW()
tc_HelloW2()
Problems
Console
Quick Access

```
// TTCN-3 version of "Hello, world!"
module MyExample
{
type port PCOType message
{
  inout charstring;
}

type component MTCType
{
  port PCOType MyPCO_PT;
}

testcase tc_HelloW() runs on MTCType system MTCType
{
  map(mtc:MyPCO_PT, system:MyPCO_PT);
  MyPCO_PT.send("Hello, world!");
  setverdict(pass);
}

testcase tc_HelloW2() runs on MTCType system MTCType
{
  timer TL_T := 15.0;
  map(mtc:MyPCO_PT, system:MyPCO_PT);
  MyPCO_PT.send("Hello, world!");
  TL_T.start;
  alt {
    [] MyPCO_PT.receive("Hello, TTCN-3!") { TL_T.stop; setverdict(pass); }
    [] TL_T.timeout { setverdict(inconc); }
    [] MyPCO_PT.receive { TL_T.stop; setverdict(fail); }
  }
}

control
```

```
TITAN console
Operation finished successfully.
*****************************************************************
2017-05-19_22:18:14: starting to build HelloWorld
*****************************************************************
sh -c make dep
make: Nothing to be done for 'dep'.
Operation finished successfully.
sh -c make all  -j 8
```

Figure 5.2: Titan TTCN-3 Workspace Environment

The attempt has been to perform the comparison between the toolsets on a common platform keeping other things constant and trying to determine the merits of using TTCN-3 to test suite development and testing as an alternative to the use of SoapUI. The same XML or JSON HTTP request messages are used by both the toolsets and similar HTTP responses are processed by both tools. Titan TTCN-3 is an implementation of the ETSI TTCN-3 standards by Ericsson, which it has recently released into the public domain as an open source Eclipse project and still used extensively by Ericsson for its telecommunications related testing. Titan TTCN-3 is a native Linux toolset. Since I used a Windows 7 PC system for this dissertation, there was a need to install a Unix/Linux emulation environment, Cygwin, over which the Titan toolset has been installed. This toolset plugs into the Eclipse IDE framework to provide a seamless development and execution environment to a test developer. Titan TTCN-3 has been used in this dissertation in a minimalistic mode, however it has many additional packages and plugins which provide graphical reports, message sequence diagrams, a whole range of pre-built adapters for a wide range of testing needs. This tool provides all the powers of a full-fledged test specification language. The SoapUI environment is developed by Smart Bear and its minimal version is released in open source community edition, whereas for more scalable and advanced features a Pro version is available in commercial form.


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Projects
CurrencyConvertor
CurrencyConvertorSoap
ConversionRate
Request 1
CurrencyConvertorSoap12
REST Project 1
REST User Mgmt
Sample Amazon Project
Sample Google Maps Project
Sample REST Project
Sample REST Project
footballpool
usaddressverification
Request 1
Inspector
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0)
Headers (10) Attachments (0) SSL Info WSS (0) JMS (0)
SoapUI log http log jetty log error log wsrm log memory log
Properties

http://www.webservicex.net/CurrencyConvertor.asmx

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envel
   <soapenv:Header/>
   <soapenv:Body>
      <web:ConversionRate>
         <web:FromCurrency>GBP</web:FromCurrency>
         <web:ToCurrency>GBP</web:ToCurrency>
      </web:ConversionRate>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="htt
   <soap:Body>
      <ConversionRateResponse xmlns="http://www.webserviceX.NET/">
         <ConversionRateResult>-1</ConversionRateResult>
      </ConversionRateResponse>
   </soap:Body>
</soap:Envelope>
```

response time: 1928ms (378 bytes)

Request Properties

Property	Value
Name	Request 1
Description	
Message Size	353
Encoding	UTF-8
Endpoint	http://www.webse...

Figure 5.3: SoapUI Workspace Environment

This environment is more command driven than programming driven and therefore less flexible than the Titan TTCN-3 environment. It provides the user with a set of Web Service request templates which can be used to draft specific RESTful client requests for instance, but these templates also do constraint the tester with bounds of flexibility whereas the TTCN-3 specification language is more of a test programming language in which rich RESTful requests can be formulated by the test developer as per their requirements, making their test cases and therefore test suites much richer.

5.1 EXPERIMENTAL RESULTS

The first set of experiments performed after the development of a common test suite in both Titan TTCN-3 and SoapUI against a common User Management RESTful Web Service (System Under Test), was to perform a comparison of the test execution performance using both the toolsets. In order to collect the execution performance timings of test cases the following detailed method was used:

T_i (<test case>) = Time taken for executing the i’th run of <test case>

The T_i value indicates the time taken by the i’th run of the test case being considered. For each test case executed against the system under test a record of the end time timestamp and the start time timestamp for the test case is made from the execution logs of the respective toolset used and using the difference between the end time and the start time the time taken for executing each run i.e. T_i (<test case>) is derived.

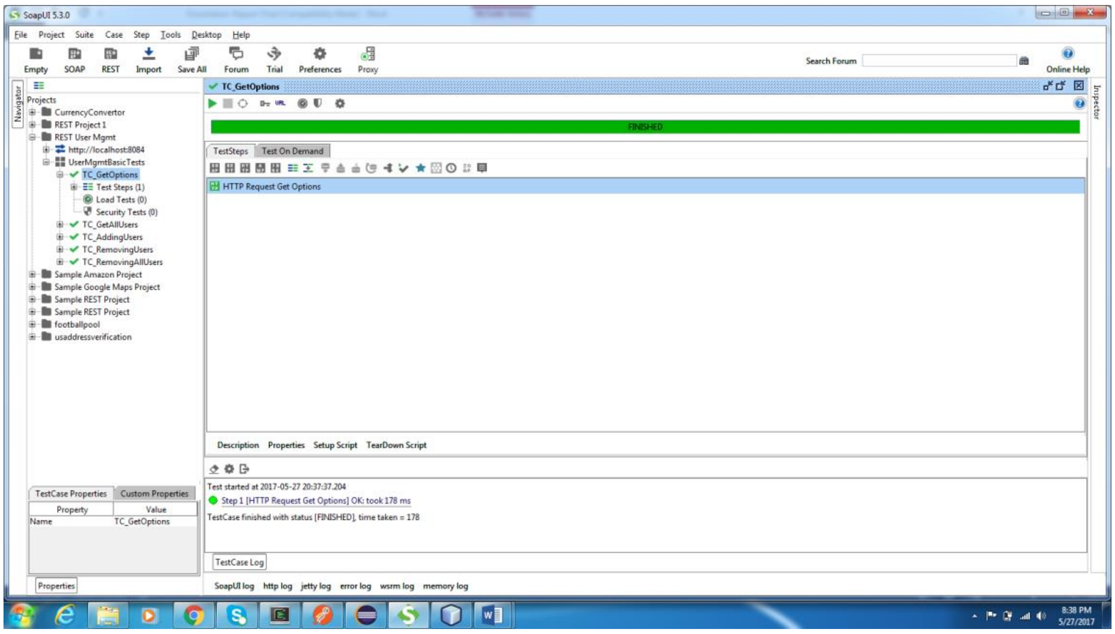
T_i (<test case>) = End Timestamp of T_i – Start Timestamp of T_i

In order to get a measure of the average execution performance of a test case with a particular toolset the following method was used to arrive at the average test case execution timings.

T_{avg} (<test case>) = Sum (T_1(<test case), T_2(<test case), T_3(<test case), T_4(<test case>), T_5(<test case)) / 5

As seen above the average is arrived at in this case by taking into consideration 5 test execution runs of each test case using both the tools. The T_i values are added together and divided by the total test runs performed for the test case which was usually five.

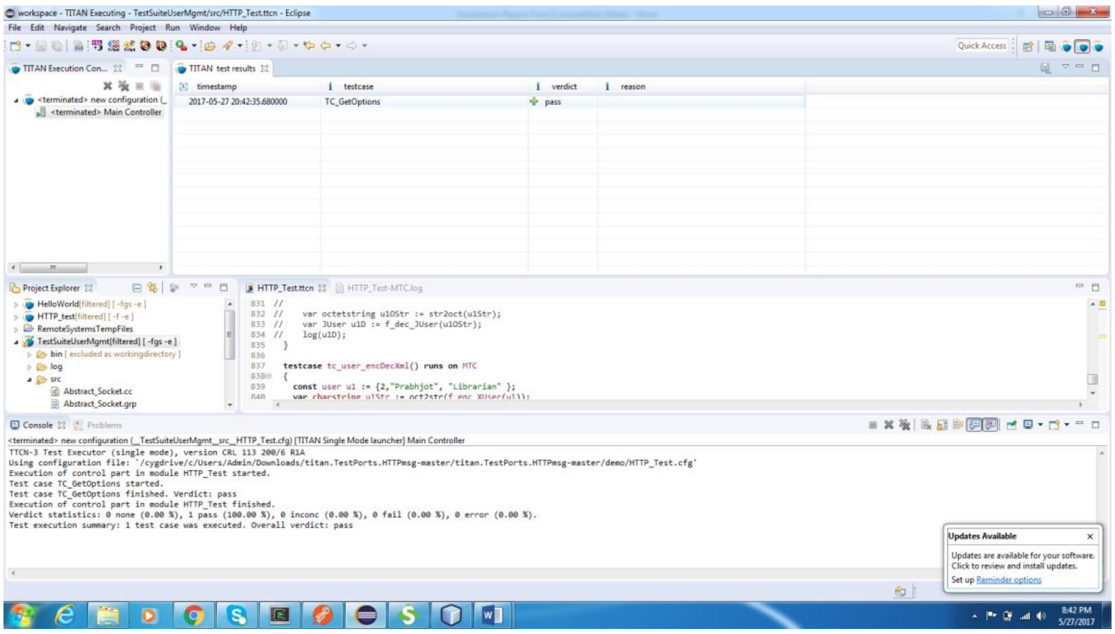
In the following two figures images of the execution logs of both the toolsets are displayed in order to clarify the collection method for timestamp data related to the execution of test cases using both the toolsets.



```
Test started at 2017-05-27 20:37:37.204
Step 1 [HTTP Request Get Options] OK: took 178 ms
TestCase finished with status [FINISHED], time taken = 178
```

Figure 5.4: SoapUI Execution View with Logs View

In the above image one can notice the test case execution duration as being 178ms for the GetAllOptions() test case executed with SoapUI. These timings are reported in the bottom right window panel of the interface with log records related to the current test case execution.

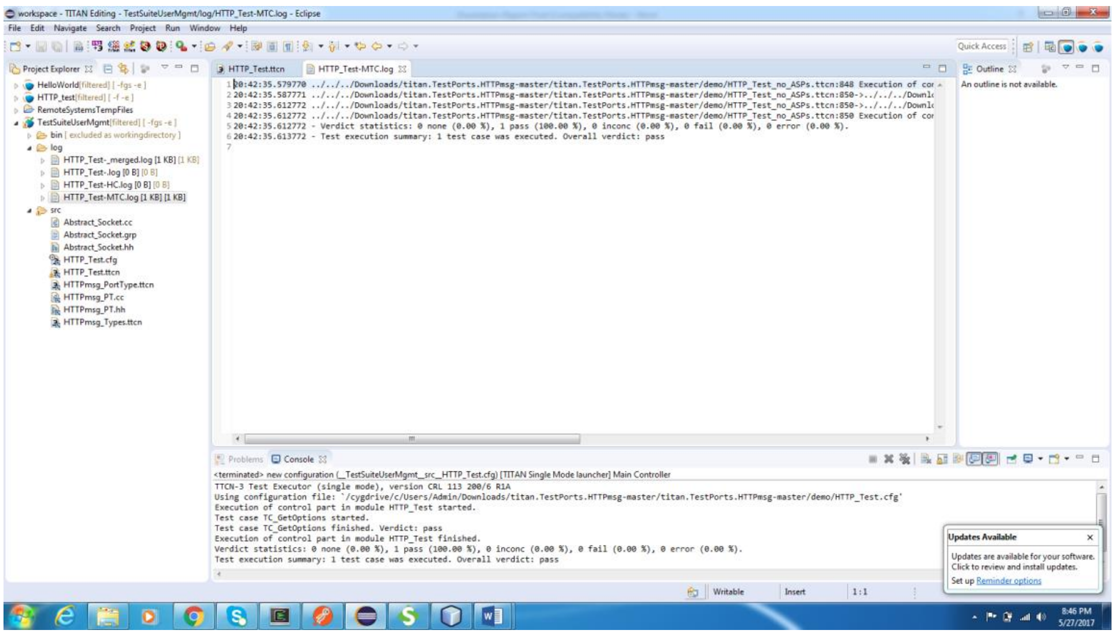


timestamp	testcase	verdict	reason
2017-05-27 20:42:35.680000	TC_GetOptions	pass	

```
//
//    var octetstring u1OStr := str2oct(u1Str);
//    var JUser u1D := f_dec_JUser(u1OStr);
//    log(u1D);
  }

testcase tc_user_encDecXml() runs on MTC
{
  const user u1 := {2,"Prabhjot", "Librarian" };
  var charstring u1Str := oct2str(f_enc_XUser(u1));
```

```
<terminated> new configuration (_TestSuiteUserMgmt_src_HTTP_Test.cfg) [TITAN Single Mode launcher] Main Controller
TTCN-3 Test Executor (single mode), version CRL 113 200/6 R1A
Using configuration file: `/cygdrive/c/Users/Admin/Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test.cfg'
Execution of control part in module HTTP_Test started.
Test case TC_GetOptions started.
Test case TC_GetOptions finished. Verdict: pass
Execution of control part in module HTTP_Test finished.
Verdict statistics: 0 none (0.00 %), 1 pass (100.00 %), 0 inconc (0.00 %), 0 fail (0.00 %), 0 error (0.00 %).
Test execution summary: 1 test case was executed. Overall verdict: pass
```

Figure 5.5: Titan TTCN-3 Execution View with Summary Logs



```
20:42:35.579770 ../../../Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test_no_ASPs.ttcn:848 Execution of cor
20:42:35.587771 ../../../Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test_no_ASPs.ttcn:850->../../../Downlo
20:42:35.612772 ../../../Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test_no_ASPs.ttcn:850->../../../Downlo
20:42:35.612772 ../../../Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test_no_ASPs.ttcn:850 Execution of cor
20:42:35.612772 - Verdict statistics: 0 none (0.00 %), 1 pass (100.00 %), 0 inconc (0.00 %), 0 fail (0.00 %), 0 error (0.00 %).
20:42:35.613772 - Test execution summary: 1 test case was executed. Overall verdict: pass
```

```
<terminated> new configuration (__TestSuiteUserMgmt__src__HTTP_Test.cfg) [TITAN Single Mode launcher] Main Controller
TTCN-3 Test Executor (single mode), version CRL 113 200/6 R1A
Using configuration file: `/cygdrive/c/Users/Admin/Downloads/titan.TestPorts.HTTPmsg-master/titan.TestPorts.HTTPmsg-master/demo/HTTP_Test.cfg'
Execution of control part in module HTTP_Test started.
Test case TC_GetOptions started.
Test case TC_GetOptions finished. Verdict: pass
Execution of control part in module HTTP_Test finished.
Verdict statistics: 0 none (0.00 %), 1 pass (100.00 %), 0 inconc (0.00 %), 0 fail (0.00 %), 0 error (0.00 %).
Test execution summary: 1 test case was executed. Overall verdict: pass
```

Figure 5.6: Titan TTCN-3 Execution View with Detailed Logs

In the two figures above are shown the test execution logs view of the Titan TTCN-3 tool, both the summary view and the detailed logs view. The execution time in this case for the GetAllOptions() test case is 25ms (end time – start time). In this manner is calculated the execution timing values of each text execution of a test case or a set of test cases from the readings that the Titan TTCN-3 and SoapUI toolsets provide in their test execution logging mechanisms. I used this same method in all of the test results that are reported in the following sections using a tabular format.

Table 5.1: Test case 'GetAllOptions' execution times

Input	Output	SoapUI Time Taken	Titan TTCN-3 Time Taken (End-Start)=
GetAllOptions	<operations>GET, PUT, POST, DELETE</operations>	170	(200-172)=28
		58	(424-405)=19
		52	(363-346)=17
		34	(546-529)=17
		30	(942-926)=16
	Average Time In Milliseconds	68.8	19.4

In the above and these following paragraphs the execution performance results are presented for both toolsets for other common test cases developed on both. The first test case was a simple 'GetAllOptions' call on a RESTful Web Service for getting the list of all methods ('OPTIONS') supported by the same. It is clearly noticed that the Titan TTCN-3 test case executes faster than the SoapUI test case in all the test runs executed (from Table 5.1).

The second common test case used was that of 'GetAllUsers'. This method would return a collection of users in XML or JSON (as requested) format containing all the users (along with their attributes) as currently being maintained by the User Management RESTful system under test. As shown in the comparison below the Titan TTCN-3 tool executes the test case in smaller amounts of time than SoapUI does. The difference between the two grows upon adding more users to be listed by the test case i.e. increasing the number of total users in the User Management module.

Table 5.2: Test case 'GetAllUsers' execution times

Input	Output	SoapUI Time Taken	Titan TTCN-3 Time Taken (End-Start)=
GetAllUsers	<List of System Users in XML/JSON>	999	(7.641-6.741)=900
		567	(7.321-6.935)=386
		492	(376-005)=371
		1987	(4.535-3.916)=619
		707	(3.360-2.860)=500
	Average Time In Milliseconds	950.4	555.2

The common test cases developed in Titan TTCN-3 and SoapUI were executed using the same platform under similar conditions within a short period of time using random ordering between the test cases being selected for execution. The third set of test cases deal with adding new users to the User Management REST system under test using list of users either drafted in XML or JSON format. The SoapUI toolset allows the user to submit the list of users in both XML and JSON format. The Titan TTCN-3 toolset lets the user submit requests in XML, JSON and other formats also. One can write and use custom encoders and decoders also in Titan TTCN-3 to support any other data format required for communicating with a RESTful Web Service. It is observed that in these test case executions also the Titan TTCN-3 toolset performs better as shown in the table below. It is also observed that as the set of users to add grows (i.e. the input data set) the Titan TTCN-3 toolset gets better at doing the job than SoapUI. This indicates better scalability potential in the Titan TTCN-3 toolset when it comes to increasing the size of test data. It can be used for a variety of scenarios such as

- Valid, invalid and inopportune testing
- Software module, unit, layer, protocol, integration and laboratory testing,
- Functional, load, distributed and testing
- Regression, certification and approval testing

Table 5.3: Test case 'AddNewUsers' execution times

Input	Output	SoapUI Time Taken	Titan TTCN-3 Time Taken
AddNewUsers	<Status Code SUCCESS/FAILURE in XML/JSON>	5433	(3.221-2.973)=248
<List of Users in XML/JSON>		461	(237-106)=131
		1008	(849-205)=644
		288	(905-812)=93
		230	(971-921)=50
	Average Time In Milliseconds	1484	233.2

These results indicate a clear advantage of using Titan TTCN-3 especially for intermediate or large scaled test systems. Titan TTCN-3 provides a distributed test execution environment which can be developed in the central IDE and then executed using a distributed components mechanism. This gives Titan TTCN-3 a very clear advantage when performing scalability tests using distributed test components. The next set of common test cases used for performance comparison check the agility of the tools while executing calls to delete existing set of users from the RESTful system under test. The fourth set of common test cases involved removing a list of users (in XML or JSON format) from the User Management RESTful system under test. As expected the TTCN-3 toolset performs better in terms of shorter time to execute the removal test cases. Titan TTCN-3 has the added advantage of programmatically generating test users and generating XML or JSON on the fly rather than editing that is required with the SoapUI interface. This is a point that really differentiates the capabilities of Titan TTCN-3 from the SoapUI toolset which would be usable only for small test suites. While Titan TTCN-3 can scale to a large set of auto-generated test data that could be used for testing either from a single node or multiple nodes against a common system under test.

Table 5.4: Test case 'RemoveSpecificUsers' execution times

Input	Output	SoapUI Time Taken	Titan TTCN-3 Time Taken
RemoveUsers	<Status Code SUCCESS/FAILURE in XML/JSON>	123	(132-83)=49
<List of Users in XML/JSON>		301	(212-170)=42
		72	(811-774)=37
		98	(680-580)=100
		125	(781-695)=86
	Average Time In Milliseconds	143.8	62.8

The test cases were composed in various combinations giving results similar to what is shared in the earlier performance comparison tables.

5.2 Comparison Results

These results lead to a confirmation of one of the dissertation objectives in terms of saving time in test execution so as to enable shorter testing cycles and quicker ready to market timelines especially for intermediate or large scale Web Service product testing environments. In commercial settings with a large scale system there are potentially hundreds and thousands of test cases to be executed in order to determine the current status of a system under test. It is apparent what a test developer can gain by writing their test system using Titan TTCN-3 so that they get the development strength of an internationally standardized testing language along with fast execution of their test systems against the SUT. When drawing average execution speed figures the averages from all test runs was compiled for Titan TTCN-3 and SoapUI respectively, for the test cases I developed and executed with, using the computation below:

SoapUI: T_{avg} (661.75) = Sum (T_{1avg}(68.8), T_{2avg}(950.4), T_{3avg}(1484), T_{4avg}(143.8)) / 4

TTCN-3: T_{avg} (217.65) = Sum (T_{1avg}(19.4), T_{2avg}(555.2), T_{3avg}(233.2), T_{4avg}(62.8)) / 4

Giving the following results in graphical form:

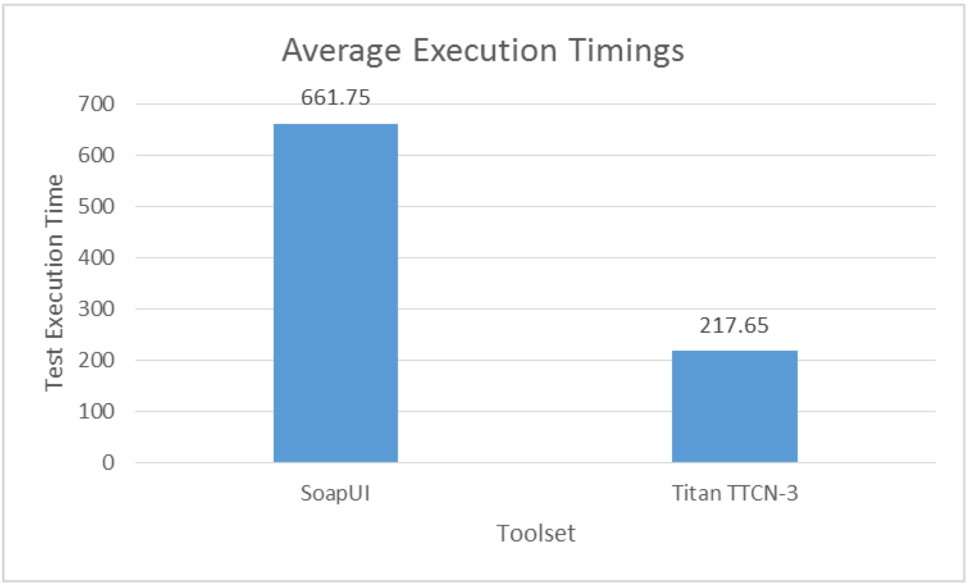


Figure 5.7: Averaged Execution Times across all runs

From the above bar graph it is clear that for our comparative test suites the TTCN-3 test cases take almost less than half the time that the corresponding SoapUI test cases take to execute against the common set of RESTful service methods. I have looked at results of test execution for both the open source tools and Titan TTCN-3 performs better and faster when executing a test system against an SUT than SoapUI.

Now in order to consider the other aspect of the comparison which involves the cost benefits of using Titan TTCN-3 over popular tools such as SoapUI, it is brought to notice that Titan TTCN-3 is free software in public domain whose wide set of capabilities with all bells and whistles potentially provide scalability and performance gains over expensive commercial toolsets such as SoapUI Pro, especially in technically savvy test environments, with test developers skilled to undertake the task of developing powerful and robust test systems and suites for the complex systems under test in their test environments.

Using reference of comparisons by organizations [30, 33, and 34], the following benefits for use of TTCN-3 testing technology are listed:

Less Time and Costs

- Reduction of development time for new testing platforms by 20 to 30 percent
- Savings of 30 to 50 percent in the adaptation and maintenance of test suites
- Low investments for training and knowledge transfer

Quality

- Testing in early design stages of system development
- Systematic, automated test methods with tool support

Well Established on the Market

- Broad range of vendor-independent TTCN-3 tools available
- Manufacturers - Motorola, Siemens, Nokia, and many others
- Carriers - Vodafone, O2, and many others globally
- Test Devices - Tektronix, Agilent, Aeroflex, Rohde & Schwarz, ...

Secure Investments

- Repeatability and continuous development
- Wide area of application test support and common methodology on a standardized level (ETSI)

Multi-Purpose Testing Language

- All kinds of black-box testing (conformance, performance, interoperability etc.)
- Development of technology-independent test suites
- Suited for a wide range of application areas and domains

User-Friendly Handling

- Easy graphical specification of test cases with TM standards
- Full test execution control on test case and test suite level
- Clear visualization of complex test scenarios with MSC's etc.
- Clearly structured test documentation with automation

Highest Flexibility in Designing and Maintaining Test Software

- Specification in various presentation formats (textual, graphical, tabular)
- Support of automated and distributed testing
- Creation of dynamic test configurations, even when distributed
- Same set of test functionalities usable in different contexts
- Reusability of existing code structure
- Easy enhancement of test data and customizations

Simple Test Adaptation

- Easy adaptation of existing test suites to newer SUT's
- Easy implementation into existing systems via standardized interfaces (TRI/TCI)
- Adaptation of generated code can be reused with SUT's

Global Standard

- The only internationally standardized test technology (ETSI MTS)
- Specifically designed for testing and used for testing
- Maintenance and continuous enhancements guaranteed
- Fast access to already standardized TTCN-2/TTCN-3 test suites

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Over the past few years many Web Services testing techniques have been developed and used, each having its own set of advantages and disadvantages. The practicing test developer is on the lookout for ways to make his job easier through automation of the testing process, so as to make the test developer focus on writing the test logic rather than worrying about the toolset limitations. In this dissertation work a test automation technique using Titan TTCN-3 is proposed which makes the Web Services testing job more effective and efficient for a test developer in the market today. The comparison methods used have brought out the strengths of TTCN-3 as a wider scoped testing language which has the ability to adapt to a diverse set of testing needs. The scope of customization that a test developer gains with TTCN-3 is quiet astounding for their testing environment. All this power is available in the open source edition of Titan TTCN-3. The comparison with SoapUI really served as a benchmark for TTCN-3 to prove itself useful for testing of RESTful web services and gladly TTCN-3 lived up to the expectations drawn from the research papers of earlier authors.

6.2 Future Scope

In this dissertation work study and observations were focused on many aspects of Web Services testing, as well as on the features of TTCN-3, which would be topics in themselves for further exploration into test techniques. For example, if one considers the details of the data template matching mechanism in TTCN-3 and how it could be well utilized for parsing and analysing web responses, which is another focus of research. The TTCN-3 toolset features could be applied in a richer manner to pursue security testing of Web Services, in fact a whole network of web services. In the near future the intention is to examine the efficiency of using Titan TTCN-3 in a scalable distributed mode with parallel executing components for testing a distributed system of Web Services. The addition of test data generation and reporting toolsets is another area of study. Test Automation with TTCN-3 is a growing area of study.

REFERENCES

[1] Willcock Colin, Deiß Thomas, Tobies Stephan, Keil Stefan, Engler Federico, Schulz Stephan, and Wiles Anthony (2011) “An Introduction to TTCN-3”, John Wiley and Sons, Germany.

[2] ETSI (2014) “TTCN-3: Core Language Specifications”, ETSI Document Number ES 201 873-14.8.1, ETSI Version 7.0, Sophia Antipolis, France.

[3] ETSI (2014) “TTCN-3: Operational Semantics Specifications”, ETSI Document Number ES 201 873-44.5.1, ETSI Version 7.0, Sophia Antipolis, France.

[4] ETSI (2014) “TTCN-3: Runtime Interface Specifications”, ETSI Document Number ES 201 873-54.7.1, ETSI Version 7.0, Sophia Antipolis, France.

[5] ETSI (2014) “TTCN-3: Control Interface Specifications”, ETSI Document Number ES 201 873-64.8.1, ETSI Version 7.0, Sophia Antipolis, France.

[6] Atkinson Colin, Barth Florian, Brenner Daniel, and Schumacher Marcus (2010) “*Testing Web-Services Using Test Sheets”*, Chair of Software Engineering, University of Mannheim.

[7] Huang Teng, Ding Zhizhong, Duan Younan, Chen Yin, and Ding Lu (2010) “*A Simplified and Efficient LTE RRC Conformance Testing Adapter”*, Institute of Communications and Information Systems, Department of Communication Engineering, University of Technology, Hefei, China.

[8] Katara, Mika (2008) “*General Purpose SUT Adapter for TTCN-3”*, Tampere University Of Technology, Department of Information Technology, Florida.

[9] Kumar Dinesh (2012) “*Validation of Internet Application: Study, Analysis and Evaluation”*, Shri Siddhi Vinayak Institute of Technology, Bareilly, Int.Journal Advanced Networking and Applications.

[10] Liu Yang and Hang Bei (2009) “*TTCN-3 Test Adapter Framework with capability Description”*, University of China, China.

[11] Rentea Cosmin, Schieferdecker Ina, and Cristea Valentin (2011) “*Ensuring Quality of Web Applications by Client-Side Testing Using TTCN-3”*, Fraunhofer FOKUS Institute, Berlin, Politechnica University, Bucharest.

[12] Schieferdecker Ina and Stepien Bernard (2008) “ *Automated Testing of XML/SOAP based Web Services”,* FOKUS, Berlin, Germany, University of

Ottawa, Canada.

[13] Schieferdecker Ina and Vassiliou-Gioles Theofanis (2004) “*Tool Supported Test Frameworks in TTCN-3”*, Fraunhofer Research Institute for Open Communication Systems (FOKUS), Testing Technologies IST GmbH, Berlin, Germany.

[14] Schieferdecker Ina, Din George, and Apostolidis Dimitrios (2005) “*Distributed Functional and Load Tests for Web Services”*, Technical University Berlin/Fraunhofer FOKUS/Testing Technologies, Berlin, Germany.

[15] Schulz Stephan (2011) “*Test Suite Development with TTCN-3 libraries”*, European Telecommunication Standards Institute, Published by Springer-Verlag 2008.

[16] Stepien Bernard and Peyton Liam (2014) *“Innovation and evolution in integrated Web Application Testing with TTCN-3”*, European Telecommunication Standards Institute, Published by Springer-Verlag.

[17] Stepien Bernard, Peyton Liam, and Xiong Pulei (2008) “*Framework Testing of Web Applications using TTCN-3”*, European Telecommunication Standards Institute, Springer-Verlag.

[18] Stepien Bernard, Peyton Liam, and Xiong Pulei (2012) “*Using TTCN-3 as a Modeling Language for Web Penetration Testing”*, University of Ottawa.

[19] Sun M, Zou J, and Ma Shi H (2010) “*Automatic generation of Codec's for a TTCN-3 Test Suite”*, Communication Software and Networks, ICCSN'10.

[20] Werner Edith, Grabowski Jens, Troschz Stefan, and Zeiss Benjamin (2008) “*A TTCN-3 based Web Service Test Framework”*, Software Engineering for Distributed Systems Group, Institute for Computer Science, University of Gottingen, Germany.

[21] Yanwu Tang (2010) “*A Versatile Codec Library (VCL) for TTCN-3”*, University of Science and Technology, Hefei, China.

[22] Zhao Huiqun, Sun Jing, and Liu Xiaodong (2012) “*A Model Checking Based Approach to Automatic Test Suite Generation for Testing Web Services and BPEL”*, North China University of Technology, Beijing, China, Edinburgh Napier University, Edinburgh, United Kingdom.

[23] Malik Vinita, Gahlan Mamta (2013) “Comparative Study of Automated Web Testing Tools”, International Journal of Latest Trends in Engineering and Technology.

[24] Kumar Ravi, Singh AJ (2015) “A Comparative Study and Analysis of Web Service Testing Tools”, International Journal of Computer Science and Mobile Technology.

[25] Wala Tanuj, Sharma Kumar Aman (2015) “A Comparative Study of Web Service Testing Tools”, International Journal of Advanced Research in Computer Science and Software Engineering.

[26] Glenford J Myers, Badgett, Todd M Thomas, and Crey Sandler. The Art of Software Testing, Second Edition, John Wiley and Sons, Inc., 2004.

[27] Dai, Z. R., Deussen, P. H. (2005) Automatic Test Data Generation with TTCN-3 Using Classification Tree Method.

[28] Web Service Basics: ‘ https://en.wikipedia.org/wiki/Web_service ’

[29] Model Based Testing: ‘https://en.wikipedia.org/wiki/Model-based_testing’

[30] Eclipse Project for Titan TTCN-3: ‘https://projects.eclipse.org/proposals/titan’

[31] The SoapUI Testing Tool: ‘https://www.soapui.org/’

[32] The VirtualBox: ‘https://www.virtualbox.org/’

[33] TTCN-3 ETSI: ‘http://www.ttcn-3.org/’

[34] TTCN-3 at Spirent: https://www.spirent.com/go/TTCN-3/

[35] User Guide for TITAN TTCN-3 Test Executor, Ericsson 2016.

[36] Installation Guide for the TITAN TTCN-3 Test Executor, Ericsson 2016.

[37] API Technical Reference for TITAN TTCN-3 Test Executor, Ericsson 2016.

[38] Programmers’ Technical Reference Guide for the TITAN TTCN-3 Toolset, Ericsson 2016.

[39] Users Reference Guide for SoapUI Community Edition, SmartBear 2016.