

**AN ONTOLOGY-BASED APPROACH FOR TEST CASE
MANAGEMENT SYSTEM USING SEMANTIC
TECHNOLOGY**

MANSOOR ABDULLATEEF ABDULGABBER ABDULHAK

**FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR
MALAYSIA**

2013

**AN ONTOLOGY-BASED APPROACH FOR TEST CASE
MANAGEMENT SYSTEM USING SEMANTIC
TECHNOLOGY**

MANSOOR ABDULLATEEF ABDULGABBER ABDULHAK

**THESIS SUBMITTED IN FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF DOCTOR OF
PHILOSOPHY**

**FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR
MALAYSIA**

2013

UNIVERSITI MALAYA
ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: MANSOOR ABDULLATEEF (I.C/Passport No: 02064802)
ABDULGABBER ABDULHAK

Registration/Matric No: WHA060019

Name of Degree: DOCTOR OF PHILOSOPHY

Title of Project Paper/Research Report/Dissertation/Thesis (“this Work”):
AN ONTOLOGY-BASED APPROACH FOR TEST CASE MANAGEMENT
SYSTEM USING SEMANTIC TECHNOLOGY

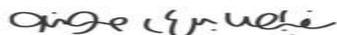
Field of Study: SOFTWARE TESTING AND SEMANTIC TECHNOLOGY

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This Work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya (“UM”), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate’s Signature

Date: Jan 2013



Subscribed and solemnly declared before,

Witness’s Signature

Date: Jan 2013

Name: Prof. Dr. Mohd Sapiyan Baba
Designation: Supervisor

To the world of Semantic Quality

Abstract

The Ontology-based Test Case Management System has been developed to maximize the use of Semantic Technology in representing and processing individual test cases for automate and reuse purpose. Effective and efficient use of test cases is desirable of any testing process. In order to achieve this an automated test case management system that is ‘knowledgeable’ is needed, where concepts and terms related to testing are important to support automated reasoning about test cases as well as for promoting common understanding among software testing practitioners involved. This thesis presents an ontology-based approach for test case management that leverages on the emerging semantic technology for developing its knowledge component. Under this approach individual test cases are structured in such a way that the important attributes, metadata, as well as linkages to related software artefacts and software testing ontology are all captured and represented using Semantic Web languages. The software testing ontology is constructed using a software testing glossary that is based on IEEE Standard as a basis. As a proof of concept an ontology-based test case management system has been developed based on this approach with the incorporation of novel features such as Automated Information Extraction and Test Case Semantic Search. The Semantic Software Testing Case Management System is found to be useful in representing and managing the Well-Structure Test Case. The thesis also discusses how the system has been validated against its objectives and argues for some perceived benefits it can bring to software testing environments.

Abstrak

Sistem Pengurusan Kes Ujian berasaskan Ontologi telah dihasilkan bagi memaksimumkan penggunaan Teknologi Semantik dalam menerangkan dan memproses kes-kes ujian yang berasingan bagi tujuan automasi dan penggunaan semula. Penggunaan kes-kes ujian secara cekap dan berkesan adalah wajar untuk apa jua proses ujian. Bagi mencapai matlamat ini, suatu sistem pengurusan kes ujian automatik yang 'berpengetahuan' diperlukan, di mana konsep dan istilah yang berkaitan dengan ujian adalah penting bagi menyokong taakulan secara automatik mengenai kes-kes ujian serta mempromosikan pemahaman umum di kalangan pengamal ujian perisian yang terlibat. Tesis ini mengemukakan satu pendekatan berasaskan ontologi bagi pengurusan kes ujian dengan memanfaatkan teknologi semantik yang sedang membangun untuk menghasilkan komponen pengetahuannya. Dengan pendekatan ini, kes-kes ujian individu distrukturkan sedemikian rupa agar ciri-ciri penting, *metadata* serta rantaian kepada artifak perisian dan perisian ujian ontologi yang berkaitan kesemuanya dirangkumkan dan diterangkan menggunakan bahasa Web Semantik. Ontologi ujian perisian dihasilkan dengan menggunakan glosari ujian perisian berdasarkan Standard IEEE. Untuk pembuktian konsep, suatu sistem pengurusan kes ujian berasaskan ontologi telah dihasilkan berdasarkan pendekatan ini dengan penggabungan ciri-ciri baru seperti Pengekstrakan Maklumat Secara Automatik dan Gelintaran Kes Ujian Semantik. Sistem Pengurusan Kes Ujian Perisian Semantik didapati amat berguna dalam menerangkan dan menguruskan Kes Ujian Tersusun. Tesis ini juga membincangkan bagaimana sistem ini telah disahkan selaras dengan objektif-objektifnya serta mempertahankan manfaat yang dianggap boleh membawa faedah kepada persekitaran ujian perisian.

Acknowledgments

First and foremost, I would like to thank the Creator of the Universe, Most Gracious and Most Merciful, without whose Will, it would not have been possible for me to fulfill my wish of completing this PhD.

I have no words to express my heartfelt gratitude to my beloved parents, Abdullateef and Asia; and my brother Abdulgabber, and sisters Noam, Shifa and Mahlia for their unconditional love, continuous support and trust in me.

My most sincere thanks to my supervisors, Prof. Sopian, for his unceasing guidance and assistance in encouraging me to grow professionally; and Prof Nor Adnan for his efforts, ideas and supervision throughout these years. I want give special appreciation to Prof. Siti Salwah and Prof. Nazim Madhavji for all their help and cooperation during my initial period as a researcher.

There are many other individuals whom I would like to thank for their direct and indirect support during the completion of my thesis, mainly Samih and Tirad for they have made the experience of doing my PhD research much more bearable. Thank you for all the wonderful memories and moments we shared together, the endless discussions during our tea breaks, the late nights spent working at the lab, and for all the constructive and positive feedback. I want to give special gratitude and affection to Samih, without whom the development of this work would not have been possible, my thanks also to Suraya for proof reading my thesis.

Last but not the least; I would like to thank the government of the Republic of Yemen for providing me with a full PhD scholarship, which has enabled me to complete my studies here in Malaysia.

Table of Contents

To the world of Semantic Quality	V
Abstract	VI
Abstrak	VII
Acknowledgments	VIII
Table of Contents	IX
List of Figures	XII
List of Tables.....	XIV
List of Abbreviations.....	XV
1.0 Introduction	1
1.1 Motivation	2
1.2 Problem Statement	3
1.3 Research Aim	5
1.4 Statement of Objectives.....	5
1.5 Research Methodology.....	7
1.6 Thesis Overview.....	9
2.0 Semantic Technology and Software Testing.....	11
2.1 Semantic Web Technology	11
2.1.1 Semantic Applications	13
2.1.2 Semantic Web Technology and Knowledge Management	15
2.2 Ontology-Based System.....	16
2.2.1 Building Ontology.....	18
2.3 Software Testing.....	25
2.3.1 Testing Concepts.....	26
2.3.2 Testing Activity.....	27
2.3.3 Testing Efforts.....	29
2.4 Software Testing Automation and Management.....	30
2.4.1 Test Case	32
2.4.2 Test Case Assessment	33
2.4.3 Test Case Elements	34
2.4.4 Test Case Management Systems	35
2.4.5 TCMS Attribute	36
2.4.6 TCMS Differing Factors	37
2.4.7 Lack of Management.....	38

2.5	Summary	39
3.0	Limitation of Test Case Management	42
3.1	Automation.....	42
3.2	Individual Test Case.....	44
3.3	Software Testing Terms	46
3.4	Search Technology.....	47
3.5	Summary	49
4.0	Ontology-based Semantic Test Case Management	50
4.1	Automated Software Testing Information Extraction	50
4.2	Representing well-structured Individual Test Case.....	55
4.2.1	Design of Well Structured Test Case:.....	55
4.2.2	Design RDFS for Test Case:.....	57
4.3	Incorporation of Software Testing Ontology	61
4.4	Integration of Semantic Search Technology	63
5.0	Designing of Software Testing Ontology.....	66
5.1	Building STO with the 101 Guide.....	68
5.2	Implementation with PROTÉGÉ 4.0.....	76
5.3	Summary	87
6.0	Implementation of Semantic Test Case Management System	88
6.1	Requirement	89
6.2	Test Case Collection.....	89
6.2.1	Test Case Template.....	90
6.2.2	Test Case Sources	92
6.3	STCMS Discussion	93
6.4	Limitation	102
6.5	Summary	103
7.0	Evaluation.....	104
7.1	Evaluation Criteria	104
7.2	Evaluation Process	105
7.2.1	Evaluation of Software Test Ontology.....	105
7.2.2	Semantic Similarity.....	108
7.2.3	Usability	110
7.2.4	Performance of Semantic Search	115
7.3	Discussion	120
7.3.1	STCMS vs. Other Web Test Case Management System:.....	120
7.3.2	Benefit of using Semantic Technology:.....	122

7.4	Summary	124
8.0	Conclusion.....	125
8.1	Findings	127
8.2	Contribution.....	129
8.3	Future Work	130
	References	131
	Appendixes.....	141
	Appendix A: Ontology Vocabulary	142
	Appendix B: STCMS Documentation	168
	Appendix C: Test Cases Data	190
	Appendix D: SUS DATA.....	216

List of Figures

Figure 2-1 Usage categories for Ontologies in Software Engineering	18
Figure 2-2 Ontology development 101 Method adapted from (Natasha & Deborah, 2001)	20
Figure 2-3 Classification of languages adapted from(Su & Ilebrekke, 2006)	21
Figure 2-4 Protégé 2000 OWL Graphic Visualization View	24
Figure 2-5 Simple Software Testing Model.....	25
Figure 2-6 Functional vs. Structural Methods.....	26
Figure 2-7 Software Testing Life Cycle adopted (Kamde, Nandavadekar, & Pawar, 2006)	28
Figure 2-8 Typical Test Case Information adopted (Jorgensen, 2008).....	35
Figure 2-9 Test case management tools VS. Factors adopted (Louridas, 2011).....	37
Figure 2-10 Number of Organization using TCMS	38
Figure 3-1 Automated Tools Model.....	43
Figure 4-1 Graph Representation of RDF Triple	52
Figure 4-2 Attributes of Test Case	55
Figure 4-3 Metadata of Test Case	56
Figure 4-4 The Well-Structure Test Case RDFS	60
Figure 4-5 Demo the Term Error with similar Terms.....	62
Figure 4-6 Logic Innovative Services of Test Case	65
Figure 5-1 STO Active Ontology Tab	66
Figure 5-2 Hierarchy Storage Test Case Suite in STO	67
Figure 5-3 General Classes View for STO	76
Figure 5-4 Sub Classes view for STO.....	77
Figure 5-5 Sub-Sub Class view of STO.....	78
Figure 5-6 Object Properties View of STO.....	81
Figure 5-7 Data Properties View of STO.....	82
Figure 5-8 Individuals' view of STO.....	83
Figure 5-9 STO Some Values From restriction	84
Figure 5-10 STO all Values From restriction	85
Figure 5-11 STO Data restriction.....	86
Figure 6-1 STCMS' Component Architecture.....	94
Figure 6-2 Create Test Case Form	95
Figure 6-3 View Test Case.....	96
Figure 6-4 Edit Test Case.....	96
Figure 6-5 Semantic Search Form for Test Cases.....	97
Figure 6-6 Search Test Case by ID	97
Figure 6-7 STO Class View	98
Figure 6-8 STO Properties View.....	99
Figure 6-9 STO Individual View	100
Figure 6-10 STO Query View.....	101
Figure 7-1 Reasoners Used to evaluate the STO	105
Figure 7-2 FaCT++ “Nothing” class shows the “no exists” of Inconsistent Class	106
Figure 7-3 Pellet reasoner shows the “no exists” of Inconsistent Class	107
Figure 7-4: GUI transforms the free-text query into the semantic representation	108
Figure 7-5 A Fragment of STO terms	109

Figure 7-6 Questionnaire Results	112
Figure 7-7 The Acceptability of SUS Score Adapted from (Bangor, Kortum, & Miller, 2008)	113
Figure 7-8 Test Case Semantic Search.....	115
Figure 7-9 STCMS' main features.....	120
Figure 7-10 The Test Case seen by a human	122
Figure 7-11 The Test Case seen by a machine.....	123
Figure 3.2-1 SWTCMS Use Cases Diagram.....	163
Figure 2-1 SWTCMS Architecture Diagram	175

List of Tables

Table 1-1 Research Methodology	7
Table 2-1 Semantic Web Technology layers description	12
Table 2-2 Framework to analyze proposed building ontology methods.....	19
Table 2-3 List of Ontology Languages	22
Table 2-4 List of Ontology Tools.....	23
Table 2-5 Total effort breakdown for projects of different sizes adopted (Louridas, 2011)	29
Table 2-6 Test Case Role in Testing Measurement	31
Table 2-7 Test Case components description.....	33
Table 3-1 List of Sample Test Management System	48
Table 4-1 Representing TestCaeDetails Data	51
Table 4-2 Representing TestCaseDetails Data in Logical Formalism	54
Table 4-3 Brief description of the Test Case Attributes	56
Table 4-4 Brief description of the Test Case Metadata.....	56
Table 4-5 Dublin Core Elements Set	57
Table 4-6 Quality Assurance Elements Set.....	57
Table 4-7 Common Elements Set	58
Table 4-8 Mapping Test Case Terms to STO Concepts	59
Table 4-9 Login Test Case description	64
Table 5-1 Questions & Answers determine STO's domain & scope.....	69
Table 5-2 Analysed Findings for Existing STO.....	70
Table 5-3 Definition and general classification of STO	71
Table 5-4 Identifying the sub and sub-sub concepts of STO terms	72
Table 5-5 Examples of Properties and their inverses.....	73
Table 5-6 Examples of Data Properties with their domain and range	74
Table 5-7 Examples of Concepts' Individuals	75
Table 5-8 STO hierarchy class rules	80
Table 5-9 STO property rules	81
Table 6-1 Test Case Template for Collecting Data.....	91
Table 7-1 Results of semantic similarity.....	110
Table 7-2 Validation Checklist	114
Table 7-3 Queries Vs General Classification.....	116
Table 7-4 Tester Search Terms Evaluation.....	117
Table 7-5 Task Testing Search Terms Evaluation	118
Table 7-6 Artefact Search Terms Evaluation.....	118
Table 7-7 Environment Search Terms Evaluation	119
Table 7-8 STCMS Vs Other Testing Tools.....	121
Table 8-1 Sections map showing where in thesis research questions answered.....	126

List of Abbreviations

Term	Definition
OWL	Ontology Web Language
TC	Test Case
TCMS	Test Case Management System
STCMS	Semantic Test Case Management System
STO	Software Testing Ontology
ST	Software Testing
SE	Software Engineering
DBMS	Database Management System
ISTQB	International Software Testing Qualifications Board
XML	Extensible Markup Language
URI	Uniform Resource Identifier
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RIF	Rule Interchange Format
SPARQL	Protocol and RDF Query Language
W3C	World Wide Web Consortium
ISBN	International Standard Book Number
DL	Description Logic
KM	Knowledge Management
KMS	Knowledge Management System
IEEE	Institute of Electrical and Electronics Engineers
SRS	Software Requirements Specification
SDD	System Design Description
RUP	Rational Unified Process
UML	Unified Modelling Language
SQL	Structure Query Language
API	Application Programming Interface
DC	Dublin Core
QA	Quality Assurance
jOWL	Plug-in JavaScript library for visualizing OWL-RDFS documents
STD	Software Test Description
GUI	Graphical User Interface
SUS	System Usability Scale

1.0 Introduction

Software testing happens to be one of the major intense activities in software engineering process. Under current software testing practices, this process also includes validation and verification of software applications. Although in principle software testing cannot prove the correctness of real world software applications, the process nevertheless can provide confidence in the quality of the software. In any testing process, the choice of test cases is fundamental to its effectiveness. For large-scale software systems the number of test cases involved can be very voluminous where an automated test case management that is intelligent and knowledgeable is desirable.

Semantic web technology lies upon a set of technology layers built on each other. These layers provide a descriptive data that can be queried by machine. Moreover, Semantic Web is being considered the future Web, which is basically formed by semantic extensions to support the data necessary for connectivity and for enhancing human-computer and computer-computer cooperation. Current and future defector standards are used to describe and reason with the data on the Web. Nevertheless, Semantic Web is an extension of the current web, which is aimed at exploiting the enormous amount of documents available in the current Web.

Hence, by using the features provided by semantic web technology, opportunities will be wide open for better management, reusability and maintenance of the test cases. Using semantic technology, which is the new trend in developing knowledge-based systems (Li, Xie, & Xu, 2011), is a promising approach to be adopted for making testing more efficient and effective. This thesis presents one such approach for test case

management which is envisaged to be crucial to the success, efficiency and effectiveness of any software testing process.

1.1 Motivation

Software testing process has become essential for the software industry and its implementation to the software development life cycle would provide us with high quality and trustworthy products (Ammann & Offutt, 2008). However, the testing process is also a challenging and costly activity. Hence, proper management through automating the process would result in minimizing human errors as well as the testing costs. This thesis focuses on the development of a test case management system that, in turn, can be incorporated into any software testing system and environment. Essentially, a test case management system is about providing support for systematic development, storing and reuse of test cases. It is obvious that, the better test cases are managed, the more efficient the time and cost of the test process would be. Moreover, proper management of the linkages between test cases and other test and software artefacts will facilitate the reuse of test cases (write once, use many).

Semantic web technology grasps a range of promises for developing efficient conceptual data represented in a formalised approach. It has shown efficient results on search engines, agents, personal desktops, knowledge management and so many other areas (Shadbolt, Hall, & Berners-Lee, 2006). Furthermore, ontology leads to knowledge reuse for sharing common terms and concepts by modelling the domain knowledge constructed with the reasoning behaviour. It is notable that a sheer amount of ontology-based systems have emerged as a mainstream application in various

domains such as knowledge management, which entails the delivery of relevant knowledge within a sufficient or required time frame (Simperl, Mochol, & Bürger, 2010).

Unfortunately, existing test case management systems are not utilizing semantic technology. Hence, with the initiation of the Semantic Web concept in the aforementioned semantic technology, opportunities for ontology-based approaches are wide open for the development of semantic test case management systems. Such systems could be considered as a sub-class of knowledge-based software testing systems that has become the dream of software testing practitioners

1.2 Problem Statement

Software testing provides a wide area for research. Today, having automated support for test management is vital in many software development projects where representational issues pertaining to test cases need to be resolved. These are explored thoroughly in this thesis since they are considered to be foundational to the development of any software testing process.

Software Testing is still largely ad hoc, expensive and unpredictably effective, and that is the reason why software-testing research is facing the challenge of automation and management. This challenge of fully automating and managing the testing process that comprehensively covers all aspects of software testing that would guarantee the improvement of its usability (Bertolino, 2007). With the advent of semantic technology, we are of the opinion that the development of effective ontology-based

semantic test case management system is achievable and this effort would give some insights on how we can further achieve the goal of having fully automated software testing systems.

Test cases play a central role in software testing in gathering both functional and non functional information that relates to the quality of the software under test. For instance, Microsoft created one million individual test cases to test the Word application (Louridas, 2011). With this amount of test cases available, we should be able to utilize the usefulness of this tremendous amount of data. Unfortunately, there has been very little focus on the reusability of these individual test cases, as most computer science researchers have only been concentrating on test suites (Miller & Voas, 2006). This under-utility of the power of individual test cases motivates us to propose a novel approach to represent individual test cases in a semantic-based environment in order to enhance their reusability as well as become more amenable to automated reasoning.

Moreover, software testing terminology lacks standardization, common identification and placement. All these lead to confusion and delay among testers. Obviously, such confusion would not only give an impact on human but also any automated software (tools) testers, and it would consequently affect production costs and time within and without (third part, outsourcing, etc.) an organization (Tauhida, Scott, & George, 2007). Herein lies the strength of building the terms in the so-called **Ontology**: it provides clarification to remove the confusion of various terms used by users to describe the same component.

1.3 Research Aim

The aim of this research is to utilize the power of individual test cases and in representing them and their relationships with other test-ware and software artefacts in a semantic test case management system so that they can be well managed and reused. Test cases on their own is not quite helpful since reasoning on them would be difficult without knowledge of how they relates to other aspects in software testing in particular and software engineering in general. It is intuitively clear that in order to support this kind of reasoning a comprehensive software testing ontology is needed.

1.4 Statement of Objectives

To achieve the aim of this research and in order to contribute our research towards the testing body of knowledge, we set objectives for the research as follows:

- **Objective 1:** To analyse and derive individual **Well-Structured Test Case** using Resource Description Framework Schema (RDFS);
 - Review different test case definitions in the literature and capture the main combination of the test case
 - Derive an individual Well-Structured Test Case based on descriptions given in sources such as IEEE standard
 - Represent the structure using Semantic Web languages

- **Objective 2:** To formalize terms for **Software Testing Ontology** and use the Ontology Web Language to represent it in such a way that it can easily be used by other automated tools, software agents and knowledge management;
 - Categorise the software testing glossary
 - Building the Software Testing Ontology
 - Capture the logical relationship between the testing terms.

- **Objective 3:** To apply the Well-Structured Test Case representation, integrated with the Software Testing Ontology, to a **semantic information retrieval mechanism** to act as a *knowledge base system* for retrieving and managing knowledge in the domain of Software Testing;
 - Utilize an existing semantic search engine to perform the semantic search for individual test cases in the proposed system.

- **Objective 4:** To evaluate the approach in a **Semantic Management Application;** under the name Semantic Test Case Management System
 - Develop Ontology-based Semantic Test Case Management System, which can serve as a useful component to any automated Software Testing System
 - Evaluate the performance of the developed system

1.5 Research Methodology

This research conducted can be explained by the following table:-

Table 1-1 Research Methodology

Method	Phase	Activity
Theoretical Research Methods	Investigation	Investigate (Articles, Papers, Journals, stat of art, interviews, conferences etc...)
Practical Research Methods	Development	Analyze visualize and design the problem and propose solution
	Evaluation	Implement & Evaluate the prototype

- **Theoretical Research Methods**

This research studies the automation and management challenges in the software-testing domain. The **Investigation Phase** sub-tasks involved are:

1. Reviewing the literature and analyzing the gap guided by the following questions to be answered:-

Q1. What do we understand about the weaknesses of the current testing – automation and management?

Q2. What is the value of individual test cases? Is there any need for a test case to be well-structured and represented individually? What type of metadata and attributes need to be considered?

2. Identifying the challenges guided by the following questions to be answered

Q3. How can we use the semantic technology for individual test case management to minimize the painstaking effort and time spent on auditing all test artefacts?

Q4. How to formulate well-known standard software testing terms in ontology to minimize the confusion that occurs among software testing practitioners? How to evaluate the reasoning of the formulated terms and the TCMS efficiency?

- **Practical Research Methods:**

In order to improve the management tool for software testing process, the **Development and Evaluation Phase** sub-tasks includes:

1. Develop a prototype test case management system which supports semantic testing information retrieval in order to show how our proposed approach is going to work based on the following:-
 - a. Functional & Non Functional Requirement gathering
 - b. Specification Designing
 - c. Implementation & Testing
2. Validate the trustworthiness of the approach based on the following:-
 - a. Precision and Recall measurement for the exactness and completeness of the search result
 - b. Evaluate the usability of the prototype for the effectiveness, efficiency and satisfaction of users
 - c. Semantic Similarity to evaluate the proximity of the matching results

1.6 Thesis Overview

Semantic Test Case Management System is a formalised approach to improve the management and automation process of testing by using efficient software test terms. This thesis consists of eight chapters, which commences with outlining the main objectives and research methodology and stating the research problem and motivations. Presenting literature reviews of semantic technology and software testing immediately follows this introduction, giving special focus to test management and ontology in Computer Science have a collection of fruitful promises. These promises reflect extracting concepts instead of mere words, enhancing the search experience in any domain knowledge, automatically matching users to whatever they are searching for, and maintaining and accessing structured data sources. These reviews also explain the costly nature of testing efforts and the existing test case management tools. After the general concepts discussed in the second chapter, the novelty of this research work is expounded on by exploring the obstacles in the testing process, the proposed solution and its implementation. Within this exploration, we present the salient features of the Ontology-based Semantic TCMS, which include extracting information and managing test cases in semantic form.

The chapter also presents the theoretical foundation and shows how the data is identified and represented with its logic in semantic layers. Furthermore, the chapter answers the “how to build ontology” question and discusses in brief the ontology-based software testing systems.

The approach is put into practice by the following two chapters where we discuss the steps followed to develop the software testing ontology. This involves the implementation of the ontology using Protégé 4.0 and the illustration on it is evaluated using built-in reasoners. Then, we demonstrate the design and limitations of the STCMS. The data collection process is also presented in this applied approach to STCMS.

To conclude this thesis, we compress the evaluation of the results and the summary of the contributions made by the research. Chapter 7 describes in detail the results achieved from the Software testing ontology, test case representation, information extraction and semantic search, which were used to evaluate the quality performance. Finally, in the last chapter, we summarize the major contributions and findings made in this thesis, followed by the limitations and a glimpse of future work.

2.0 Semantic Technology and Software Testing

2.1 Semantic Web Technology

Semantic Web is considered as the future web that provides a descriptive data that can be queried by machine (Tim, James, & Ora, 2001). The semantic is emerging technology for developing its knowledge component Semantic Web Technology has been applied in various areas such as in e-Learning in (Rathod, Prajapati, & Singh, 2012), graph query processing in (Yıldırım, Chaoji, & Zaki, 2012), cloud computing in (Husain, McGlothlin, Masud, Khan, & Thuraisingham, 2011) and recommendation system in (Mahadevan, 2012). The W3C making it available for interested parties to share the success applications to maximize the use of Semantic Technology.

The data represented in the semantic web have a well-defined meaning combined with its rules of reasoning. The Semantic is achieved by describing the meaning of the resources and supporting its reasoning using Ontology Web Language. The Semantic Web Technology lies on a set of technologies layers build on each other. These layers provide a descriptive data that can be queried by machine (Antoniou & Harmelen, 2008). This approach facilitates large scale integration and sharing of the web data. In this approach the web data is linked and connected to its resources by the Uniform Resource Identifier URIs.

The layers are described in Table 2.1 as follows:-

Table 2-1 Semantic Web Technology layers description

Layer	:	Definition
URI	:	The Uniform Resource Identifier (URI) is a string of characters for identifying an abstract or physical object or resource. URI is particularly suitable for referring to objects on the web.
XML	:	The Extensible Markup Language (XML) is a language for users to mark up content using tags to structure a web document. XML is particularly suitable for sending documents across the Web.
RDF	:	The Resource Description Framework (RDF) is a language that has XML-base syntax for representing information about resources in the web. RDF is particularly suitable for representing metadata about web sources.
RDF(S)	:	The Resource Description Framework Schema RDF(S) is a language to create vocabulary for describing the RDF resources such as classes, subclasses, and properties. RDF(S) is particularly suitable for providing modelling for the Web objects.
RIF	:	The Rule Interchange Format (RIF) is a language (under process) to give the basic rules for checking.

OWL : The Web Ontology Language (OWL) is another extension of RDF(S) for describing and sharing ontologies (more info about ontology on chapter 3). OWL is defined as three sublanguages: OWL Full, OWL DL, and OWL Lite.

SPARQL : The Protocol and RDF Query Language (SPARQL) is a special query language for express queries across diverse data sources. SPARQL is particularly suitable as the results of query can be result set or RDF graph.

2.1.1 Semantic Applications

User interface and applications layer puts the semantic technology in practice. The layer explores how the technology effects positively and improves the efficiencies by integrating to the business flow in different areas. Since the last decade, the semantic literature recorded quite number of successful semantic applications. Meanwhile, the W3C is making it available for interested parties and communities to record their success applications.

In fact, the Semantic Technology has been applied in various areas such as information publishing, data integration, e-learning, e-government, e-commerce, web-services, multimedia collection indexing etc and have different focused communities for instance e-science (Hall & O'Hara, 2009). However, Breitman, et al. (2007) claims that applications can be categorized into the following:-

- **Semantic Agent:**

Seeing that the semantic technology provides a promising communication facilities for agents to integrate with each other and perform services for end users (Hendler, 2001).

In addition, to overcome drawbacks problems of semantic technology & agents on either end will be possible in integrating them (García-Sánchez, Valencia-García, Martínez-Béjar, & Fernández-Breis, 2009).

- **Semantic Desktop:**

Seeing that the semantic technology promises the information management and metadata ontologies which make it possible to allow what so-called semantic desktop vision to become real by manage, distribute, integrated and collaborate the personal information to the web (Dengel, 2007).

- **Semantic Art:**

Seeing that the semantic technology promises the ability of conceptualizing the underlie knowledge to represent a common vocabulary to be shared between cultural heritage organizations and retrieving comprehensible data that can be applied for images to enable third parties to make an intelligent decision about the relevance of the images (Osman, Thakker, Schaefer, Leroy, & Fournier, 2007).

- **Semantic Geospatial:**

Seeing that the semantic technology promises the ability of standardizing information infrastructure, machine to machine interactions and automating the service chaining for deriving knowledge, that can lead to successful discovery, automation and integration of the geospatial data and services (Zhao et al., 2009).

2.1.2 Semantic Web Technology and Knowledge Management

There are also some successful applications of semantic technology in the knowledge management area that are related to this thesis research. The following two cases from (Antoniou & Harmelen, 2008) are exemplary.

- **Skill Finding:**

It is a feature which has been created using the semantic technology. An ontology was built to represent various types of employee skills which consist of more than 1000 categorized concepts. Through this semantic extension the knowledge management system was able to construct a skill repository of different employees with different skills located in different locations. One of the major motives for such system was to establish an electronic repository of employees' experiences and skills.

- **Think Tank Portal:**

It is a feature which has been created using the semantic technology. The domain ontology used defines the knowledge domain of the research organization knowledge domain. Through this semantic extension the knowledge management system was able to represent semantically the contents such as research topics, authors, and relations between authors and respective topics of the organization's website in several ways. One of the major motives for such system was the need to disseminate the knowledge of a virtual organization.

2.2 Ontology-Based System

Ontologies have been defined in the literature and used in the industries as well, to provide conceptual vocabularies that describe a certain domain. For instance, in Science the term ontology is used to describe semantic constructs using words meaning. Ontology-based System is an established discipline that features intelligence and insight capabilities. It delivers the most related up-to-date information in the shortest possible period of time.

Ontology-based system has emerged in the mainstream of many application domains such as: E-commerce, Medical, Chemistry and the foremost Knowledge Management (KM) system (KMS). Most strategies in KM entail the delivery of relevant knowledge at the sufficient time required. There are three types of KM Ontologies (Gómez-Pérez, Fernández-López, & Corcho, 2004):

- 1) Information Ontology, which contains generic concepts and attributes;
- 2) Domain Ontology, which is used to describe the contents;
- 3) Enterprise Ontology, which is used for the organization context description.

The term ontology was first introduced in the field of philosophy. Several fields of study have now used the term with interpretations that suite their respective interests. In philosophy, the term ontology answered few questions concerned by the Greeks (philosophy of being). It tries to understand and distinguish the meaning of things, the changes of their status, and to classify the entities of the world (Gómez-Pérez et al., 2004). In Science the term ontology is derived from cognitive semantic or the science of being and used to describe semantic constructs using the meaning words (as

dictionary in linguistic)(Kang & Lau, 2007). We quote Gruber on defining Ontology as:
“*Ontology is an explicit specification of a conceptualization*”. (Gruber, 1993)

Ontologies should provide classes as the various concepts in the domain, relationships among these concepts, and properties as the attributes possess by the concepts (Breitman et al., 2007). Generally the intended purposes would determine their usages, and most of them are intended for re-use purposes. Ontology as a formal structure will be defined as $O = \langle C, R, I, A \rangle$ where C is a set of classes representing the domain concepts, R is sets of relations between the classes, I is sets of instances where each instance can be instance of one or more classes and can be linked to other instance by relation, and A is sets of axioms, representing a conceptualization of a specific domain. Happel & Seedorf (2006) provide a framework for classifying the usage of ontology in software engineering. In their framework they propose two dimensions (runtime and development in one side and domain and infrastructure on the other side) to classify the uses of ontology and came up with four basic areas of classification as shown in Figure 2.1 and described as follows:

- ✓ **Ontology-driven development (ODD):** Where ontologies used in development time to describe the problem domain
- ✓ **Ontology-enabled development (OED):** Where ontologies used in development time to support the development tasks
- ✓ **Ontology-based architectures (OBA):** Where ontologies used in run-time as primary artefact
- ✓ **Ontology-enabled architectures (OEA):** Where ontologies used in run-time as infrastructure support

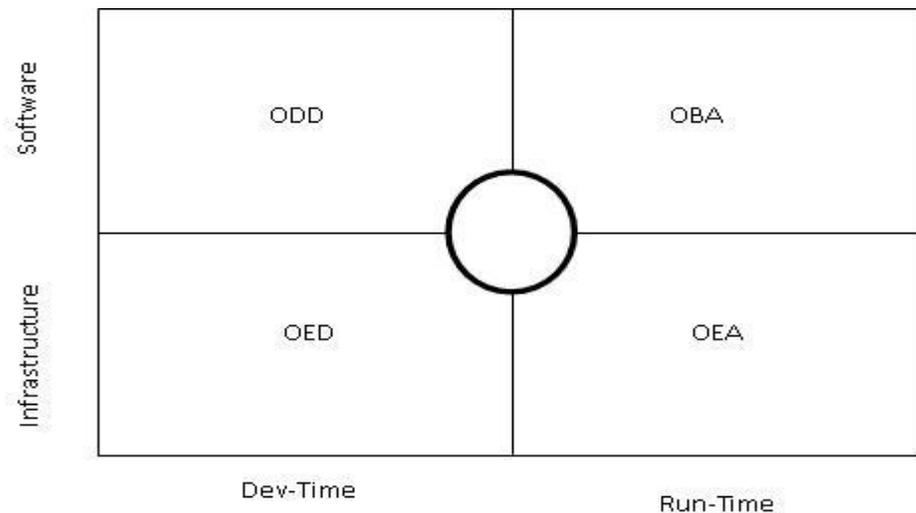


Figure 2-1 Usage categories for Ontologies in Software Engineering

2.2.1 Building Ontology

Ontology technology has reached the level of maturity by the availability of enough methodologies, tools and languages. Furthermore, ontologies are artefacts designed, formed for a purpose, and evaluated against objective criteria. The five principles for designing ontologies to be used in knowledge sharing are: clarity, coherence, extendibility, minimal encoding bias, and minimal ontological commitment (Simperl et al., 2010). Moreover, methods, languages, and tools are the main items of building up ontologies. Hence, following a comprehensive guide and using a recommended language by W3C and a stable tool will avoid what might go wrong during the runtime of the ontology.

- **Methods:**

There are no standard methods to build ontologies. Hence there are different attempts in the literature from different interest parties. Gómez-Pérez, et al. (2004) elaborated a framework to compare different methods to help users select the most useful one to

build their ontology. This framework can be used to analyze any method for building ontology. The framework provides a set of criteria and features. Table 2.2 summarize and describe their objective in short details.

Table 2-2 Framework to analyze proposed building ontology methods

Criteria	Features	Objective Description
Construction Strategy	Life Cycle Proposal	To describe activities should perform throughout the stages of ontology development.
	Strategy with respect to the application	To measure the dependency of ontology with the application using it
	Strategy to identify concepts	To determine either, bottom-up, top-down, or middle-out approach.
	Use of core ontology	To analyze the possibility of using core ontology as starting point.
Software Support	Tools that give support	To find if supported either fully or partially by tools.
Development Processes	Management Activities	To find out if management activities described and documented.
	Development Oriented Activities	To find out if pre, during and post development process are described and documented.
	Support Activities	To find out if development support activities described and documented.

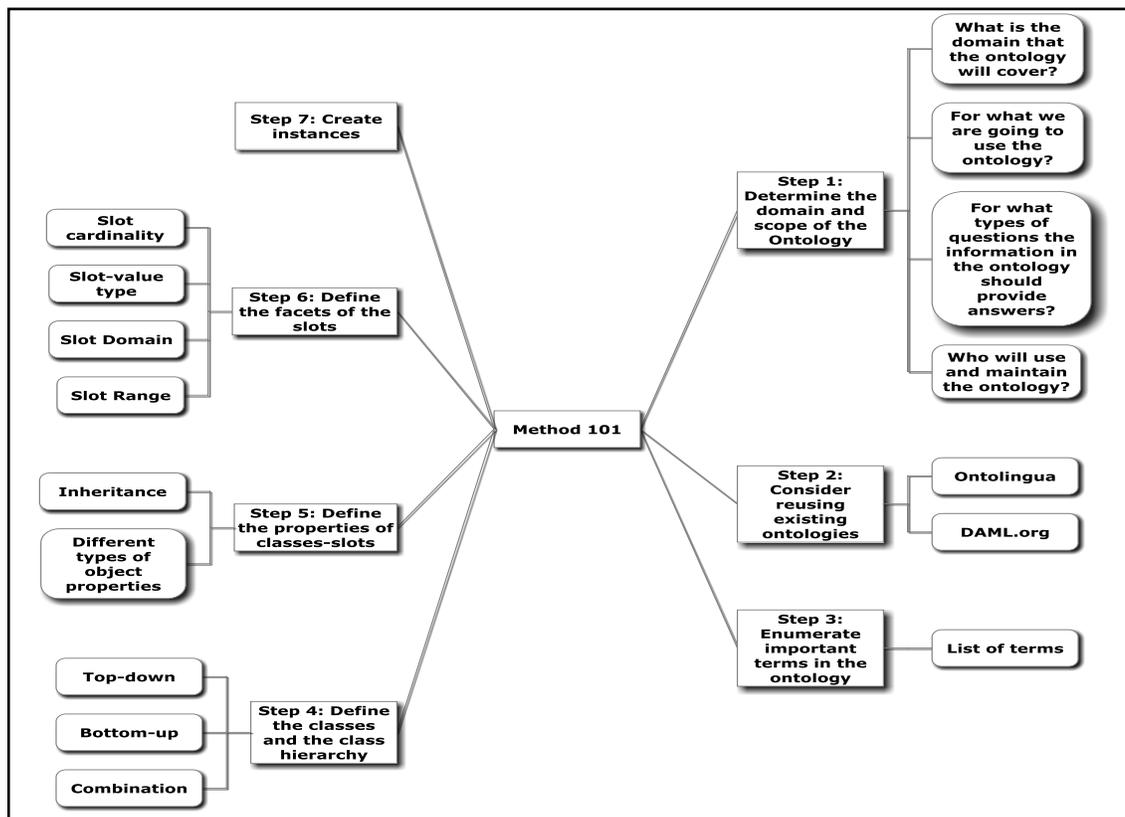


Figure 2-2 Ontology development 101 Method adapted from (Natasha & Deborah, 2001)

For the purpose of this research selection, we highlight the simplified methods proposed in (Natasha & Deborah, 2001) as a guide to create our first ontology. The authors devised the method based on their experience in using ontology-editing environment and by adopting some ontology-design ideas from the object-oriented design on literature. The method is illustrated in Figure 2.2.

In short there is no correct way to model. Constructing ontology is an iterative process that basically captures the concepts their relations in the domain of interests. There are 7 steps in the chosen method where after defining the initial version it is either evaluated by experts in the field, implemented in a case study or both.

- **Languages:**

The need of representing and exchanging data on the Internet led to the creation of web-based ontology languages. For the last few years a number of languages to support ontology in the context of what so called Semantic Web have been developed. In a summary form, Table 2.3 illustrates the most famous ontology languages. Other languages have also been used as shown in the classification of languages in Figure 2.3, traditionally, for building ontologies, but that is out of the scope of our research. The table indicates the name of the ontology, the base developed upon, reference to the developers, and purpose of developing.

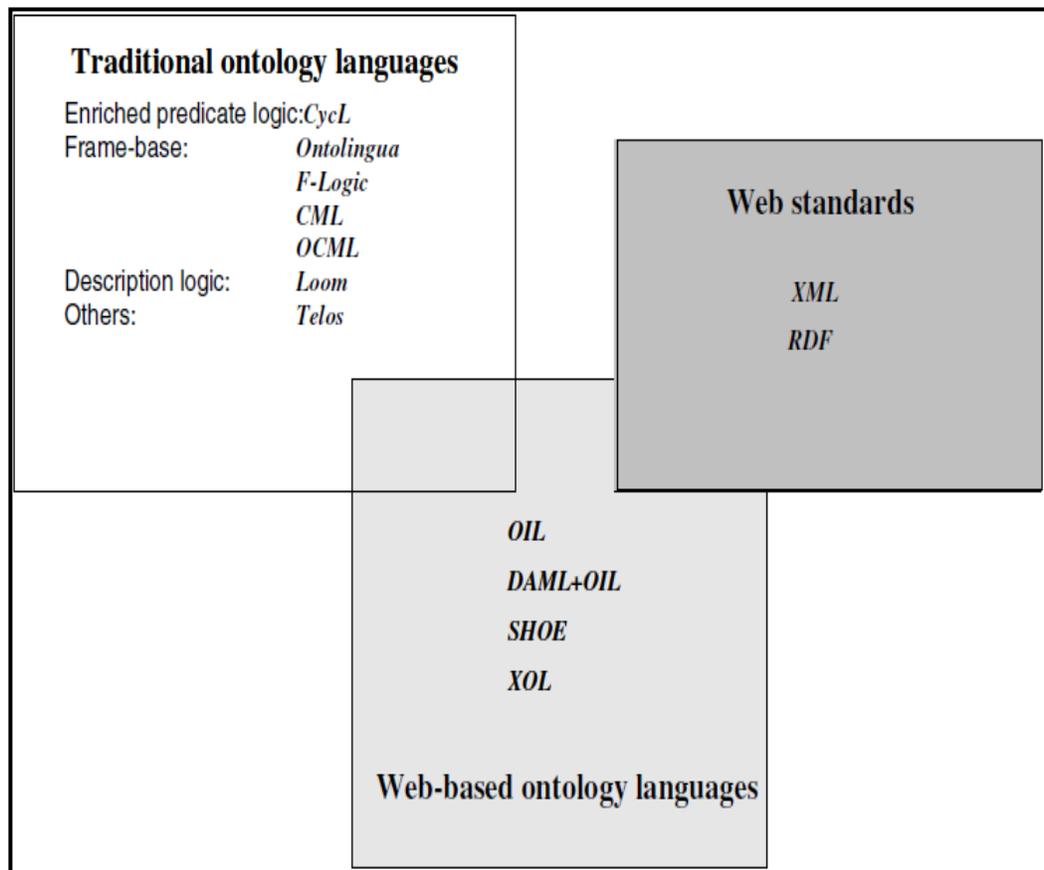


Figure 2-3 Classification of languages adapted from(Su & Iiebrekke, 2006)

Table 2-3 List of Ontology Languages

Name of Ontology Languages	Developed On	Developed By	Purpose
Ontology Exchange Language (XOL)	XML	(Karp, Chaudhri, & Thomere, 1999)	To provide a format for exchanging ontology definitions among a heterogeneous set of software systems.
Simple HTML Ontology Extension (SHOE)	HTML	(Luke S, 2000)	To improve search mechanisms on the Web by collecting meaningful information about Web pages and documents.
Ontology Inference Layer (OIL) + DARPA Agent Markup Language (DAML)	RDF(S)	(Horrocks, 2002)	To allow semantic markup of Web resources.
Web Ontology Language (OWL)	XML & RDF(S)	(McGuinness & Van Harmelen, 2004)	To publish and share ontologies in the Web

For the purpose of this research selection, we highlight in the context of Semantic Web to use the languages which are XML-based such as RDF and OWL. Among the main advantages are beside the easily of reading and managing, is the huge support from different groups and communities, which leads to the availability of more tools to edit and develop the ontology.

- **Tools:**

Building ontologies is considered as a huge complex task that requires a lot of time and manpower. Consequently, during the last decade communities and research groups build different tools aiming to facilitate the process development and the reuse of ontologies. As a result, a number of tools came to the surface with different purposes and interfaces that help users carry out their development tasks (Gómez-Pérez et al., 2004). In an ontology tools survey Perez et al.(2002) had classified tools into development tools, evaluation tools, merge and alignment tools, ontology-based annotation tools, querying tools and inference engines, and learning tools. Moreover, in a comparative study with the help of an evaluation framework, Su & Iiebrekke (2006) had found the most relevant tools to facilitate the development of ontologies. They are listed in Table 2.4 with a summary description, the name of the tool; reference to the developers, and the additional special purposes beside the editing and creating of the ontology.

Table 2-4 List of Ontology Tools

Ontology Tool	Developed by	Special Purposes
Ontolingua	(Farquhar, Fikes, & Rice, 1997)	To ease the development of Ontolingua ontologies in a shared environment between distributed groups
WebOnto	(Domingue, 1998)	To support the collaborative browsing, creation and editing of ontologies
Protégé-2000	(Noy, Fergerson, & Musen, 2000)	To support the graphical software development environment.
OilEd	(Bechhofer, Horrocks, Goble, & Stevens, 2001)	To provide consistency checking functions and automatic concept classifications
OntoEdit	(Sure et al., 2002)	To ease the development in a plug-in architecture
WebODE	(Arpírez, Corcho, Fernández-López, & Gómez-Pérez, 2003)	To support the access services by services and applications plugged in the server

For the purpose of this research selection, we look at Protégé-2000 which is an open source standalone application written in Java and provides a plug-and-play environment that specifically supports an OWL editor and reasoner. As shown in Figure 2.4 Protégé 2000 OWL plug-in provides a graphic visualization of the classes and properties using different colour codes to help developers distinguish between different types of classes (Breitman et al., 2007).

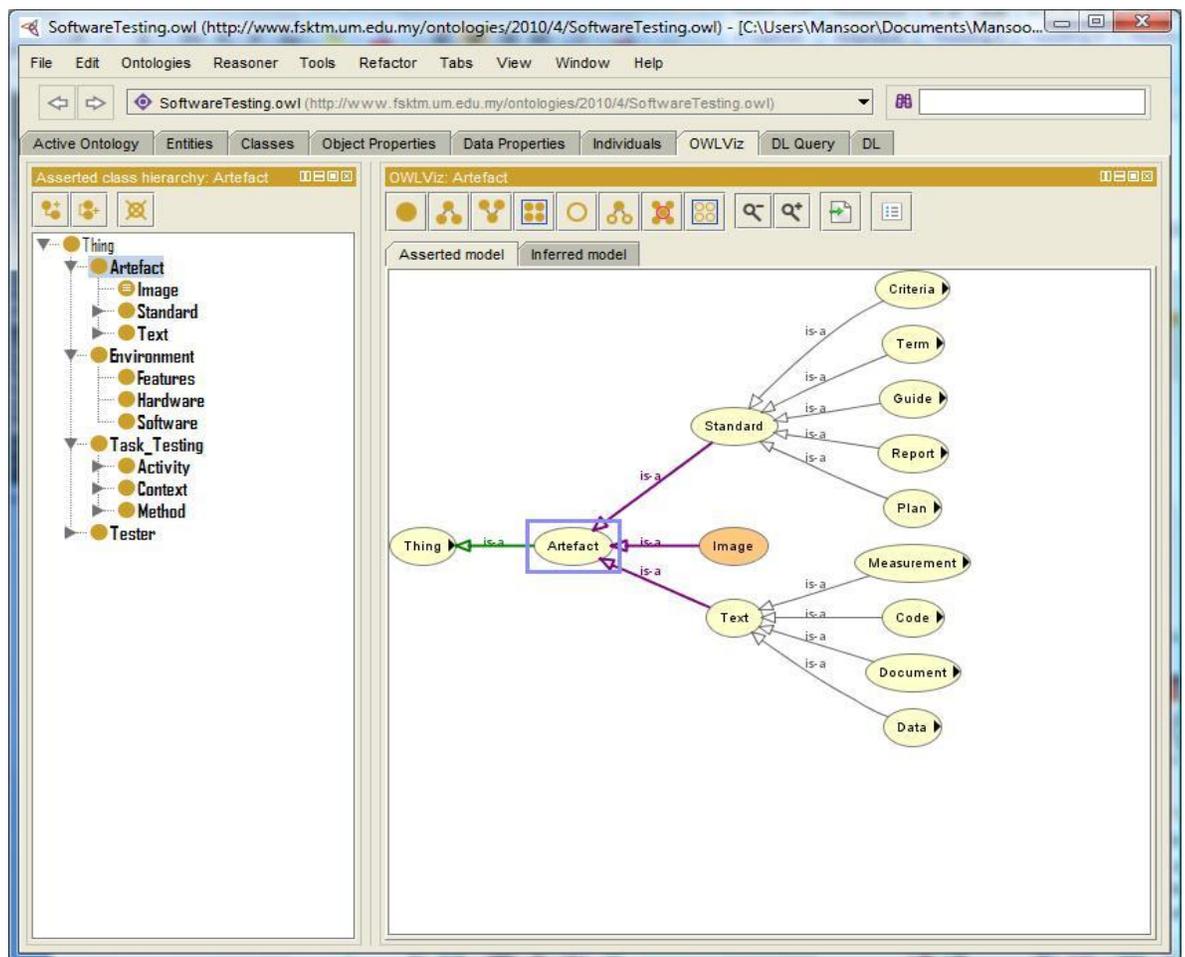


Figure 2-4 Protégé 2000 OWL Graphic Visualization View

2.3 Software Testing

The foundational philosophy of software testing as an art of finding bugs was introduced by Glenford J. Myers in 1979. When we talk about reliable software, we evidently mean a free error program. Herewith, our art falls in; to add the quality and reliability of the produced program (Myers, 2004). Software testing process is essential and important activity practiced widely in industry to ensure the quality of their products. In Figure 2.5, we show a simple Software Testing Model with the basic components of testing which are test input, system under test and the test results.

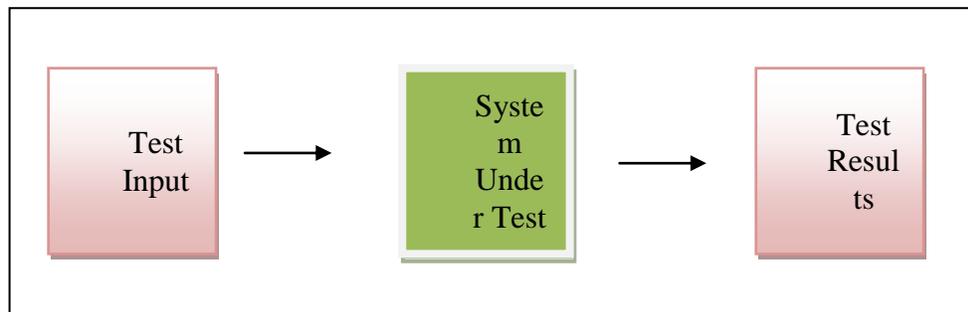


Figure 2-5 Simple Software Testing Model

Software testing is a broad area of research. It started since the beginning of computer science although it only became recognized in the middle of 70s. Research groups, professionals and practitioners from both academia and industry have been contributing to the literature with voluminous amount of research papers, books, practical reports, review papers etc (Whittaker, 2000). Despite such a progress, Bertolino (2007) argues that software testing research still faces a lot of challenges due to it being naturally unpredictably effective. To understand the importance of software testing research, it's relevant to first review the fundamental concepts of software testing.

2.3.1 Testing Concepts

Testing techniques are considered as different approaches used to perform the testing processes which include human testing techniques or mathematic testing techniques. Testing techniques are classified into static and dynamic testing. Unlike static techniques, dynamic techniques require the execution of the software. Static techniques, also known as static analysis or static code analyses, rely on reviewing and analyzing the code or other testing artefacts (Ammann & Offutt, 2008).

Two important dynamic testing techniques are black and white box testing. The purpose of the black box technique is to find out situations that the system behaves in such way it shouldn't without interfere with the internal structure of the program. Black box testing (also known as functional testing) is based on requirement and/or specification design to design the test cases. Where, the purpose of the white box method is to examine the internal structure of the program. White box testing also known as structural testing the designing of its test cases based on the implementation of the software entity. As shown in figure 2.6, structure-based testing applies the validation of the code while the functional testing is more to the system level (Heiser, 1997; Woodward & Hennell, 2004).

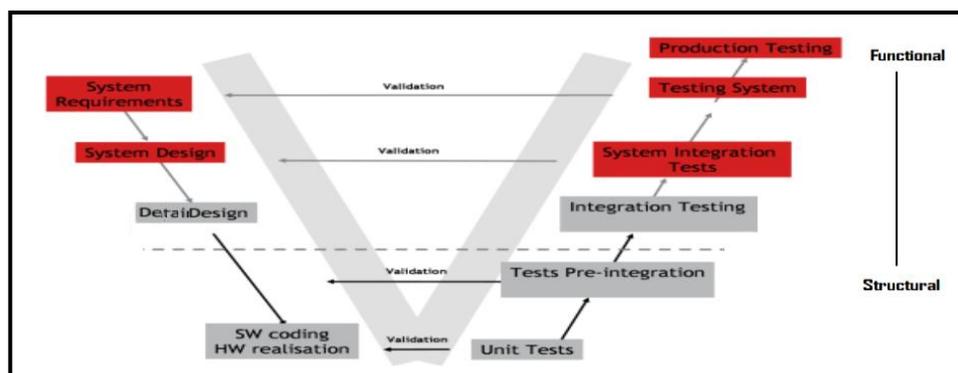


Figure 2-6 Functional vs. Structural Methods

In Beizer (2002) approach, tests are derived based on the maturity level which is characterized by the goals of test engineers. However, each test would differ in its nature and objective. Testing can be derived based on the software activities i.e. requirement, design artefact, or the source code.

2.3.2 Testing Activity

Software testing is an important process comprising of activities being practised widely in software industry to validate the software they produced. Since it provides a realistic feedback about software behaviour it can thus be viewed as an important of software quality assurance. Activities related to software testing put great emphasis on the importance evaluation in support of quality assurance through gathering information about the software under test.

Essentially software testing process should cover analysis, design, and execution of test cases as well as evaluation of the test results (Mary Jean, 2000). Furthermore, whenever a tester decides to test any program he has to also consider the environment related to the software such as the platform, source code and the interfaces. The main predicament, testing process is a challenging and costly and flaws of designing a good test cases. As well, testing is part of an overall project. Thus testing must respond to real project needs, so test projects require test project management (Rex, 2002).

In light of this understanding, we could say that testing is a wide area which involves both technical and non technical activities. In addition, it's a process that depends on

context that needs to be well managed. Figure 2.7 illustrates the testing activities in PDCA (Plan, Do, Check, and Act) steps used in management.

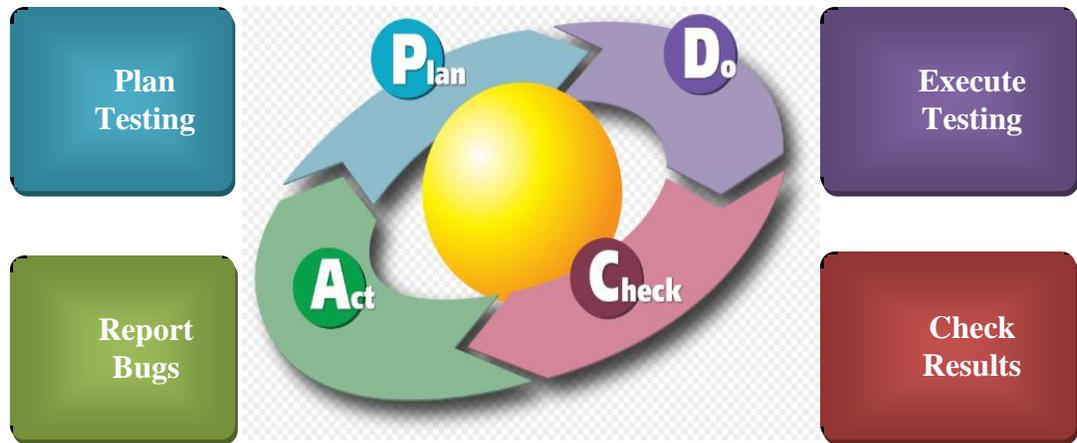


Figure 2-7 Software Testing Life Cycle adopted (Kamde, Nandavadekar, & Pawar, 2006)

✓ **Plan Testing Phase**

In this phase testers describe scope and approach of the test, schedule the testing process and identify the items need to be tested.

✓ **Execute Testing Phase**

In this phase testers develop the test cases and then run them to test the required code.

✓ **Review Results Phase**

In this phase testers review reports of actual testing results and compare them with expected test results.

✓ **Report Bugs Phase**

In this phase testers report the bugs to the development team to fix and generate matrices for the final report on whether the product can or cannot be released.

2.3.3 Testing Efforts

As we have seen with testing activities during the development of software products in the previous sub heading, the efforts of these activities is costly. Depending on the size and the nature of the software product, the testing efforts will be affected. Generally, more testing efforts are needed in security critical products that have high impact on real life. Real-time systems normally also require more testing efforts in order to validate the timing aspects of the requirements.

Table 2-5 Total effort breakdown for projects of different sizes adopted (Louridas, 2011)

KLOC	Activity				
	Requirements	Architecture & planning	Construction	System Test	Management, overheads
1	4%	10%	61%	16%	9%
25	4%	14%	49%	23%	10%
125	7%	15%	44%	23%	11%
500	8%	15%	35%	29%	13%

Table 2.5 illustrates the size of testing efforts testing relative to other software development activities and how it grows with respect to the size of the product measured in KLOC (KLOC is called as 1000 lines of code). It will require 16 to 29 percent of the total efforts of the project to perform the testing activities. Therefore, with this amount of effort, proper management of the activities will help minimize the time required and reduce the total cost of the final products. Moving beyond the activities, related concepts and efforts, the most important consideration in software testing is the test case itself (Myers, 2004).

2.4 Software Testing Automation and Management

Software testing automation is a set of concepts and tools that facilitate the testing process. There are numbers of frame work such as in (Puri, 2012) and approaches such as in (Heiskanen, Maunumaa, & Katara, 2012) have been developed to make test automation more efficient. Moreover we found some of these techniques still selecting test cases manually for instance (Kekkonen, Kanstrén, & Heikkinen, 2012). Meanwhile, in Wiklund, Eldh, Sundmark, & Lundqvist (2012) qualitative evaluation indicate that development of test automation tools encounter problems. Additionally, Rafi, Moses, Petersen, & Mantyla (2012) found that automation bares a high initial cost in designing the test cases.

Therefore, these frameworks and approaches are giving less attention to individual test case management and reusability. Actually the testing process is an extensive area involving technical and non technical activities and to perform the testing process test cases are the inputs to test the software. The efforts of these activities bare a high cost and the context of these test cases requires well management. Automated testing and testing management are critical issues in many software development projects and we quote Louridas saying: *“In many projects, testing consumes the single biggest amount of resources of all activities. We tend to collect test cases like stamps without clear strategy— just in case. Many companies suffer with insufficient quality, visibility, and test progress management.”* (Louridas, 2011)

The test case increases the quality of testing to such an extent that it becomes the most valuable component in the testing activities, not just in the central position of testing. Hence, the test case is used as the main element to measure the efficiency of the test process. If the test case is structured and developed well, the testing performance will be more accurate. Therefore, with whatever approaches is used to measure the testing somehow consider the test case is a major element for the accuracy of the testing. In Table 2.6, we show an example of the role of test case in test process efficiency measurement.

Table 2-6 Test Case Role in Testing Measurement

Measurement Approach	Role of Test Case
Defect removal efficiency	The number of Bugs found by the Test Cases to the total number of bugs found in the complete product life cycle.
Test efficiency	The number of Test Cases executed divided by time of execution and/or Test Cases executed divided by number of total Test Cases required.
Test effectiveness	The number of bugs found in a product divided by the number of Test Cases executed .
Test coverage	The number of Test Cases covered the different phase of requirements, design, code and interfaces of the product life cycle.

Understanding the purpose of test cases can assist in developing the test case itself by providing comprehensible language and standard order (Gupta & Surve, 2011). These elements affect the quality of the test case (Kamde et al., 2006). If the language used to develop test cases is vague and the attributes of the test case are disordered, the testers waste a tremendous amount of time trying to decipher the language and the order of the test case before proceeding with the evaluation. This impacts the re-usability of the test case. However, this drawback can be avoided by having a good test case management system.

2.4.1 Test Case

Software testing can improve the quality of any software by gathering information during analysis, design, and execution of test cases. The IEEE Standard Glossary of Software Engineering Terminology (1990) defines test case as “A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement”. Test cases occupy a central position in testing that has a set of input with a list of expected results that has an identity and is associated with program behaviour. Each test case defines the inputs and procedures to be tried and followed to test software. The test case can be a structural or behavioural design (Jorgensen, 2008).

Based on the above, a test case can be considered as a road map that provides the information necessary to execute the testing process. On the other hand, Ammann & Offutt (2008) claim that it is the role of a test engineer who designs the artefact since he

is in the best position to define the test cases as each of the software artefact produced should have an associated set of test cases. Table 2.7 illustrates the purpose of the main components of a test case based on the 892-1998- IEEE standard for Software Test Documentation as follows:-

Table 2-7 Test Case components description

Component		Purpose
Test Case specification identifier		Test Case ID
Test items		Brief description of the item to be tested
Input specifications		Brief description of the input values
Output specifications		Brief description of the expected output values
Environmental needs		Brief description on the testware
Special procedural requirements;		Brief description on constraints
Intercase dependencies		Brief description on the nature of dependencies

2.4.2 Test Case Assessment

Over the last decade, many professionals wrote on the art of test case engineering. Test case engineering involves designing good test cases, which can be a challenge without a systematic approach to the process. There are no secret guidelines to produce so-called good test cases. However, the purpose of the test itself determines if it results in a good

test case. Test cases are designed, in the first place, to retrieve information from the test regardless if that is pass or fail (Kaner, 2003).

We can say that to achieve test systems that are effective, efficient, integrated and maintainable, especially the testware (i.e. test case), we must develop the practice of building well-structured test cases. What underlies an effective test system is when each test case's foundation is built with proper components. Each one should consist of the test case setup to describe the steps needed to configure the test environment, the test conditions to assess the quality of the system, and the test case teardown to specify the steps needed to restore the test environment (Rex, 2002).

2.4.3 Test Case Elements

A test case comprises test case values, expected results, prefix values, and postfix values (Ammann & Offutt, 2008). Furthermore, a well developed test case would consist of the most obvious information input, expected output and management information. The input information is called precondition (the prior circumstances), and the actual input (developed by testing methods). While the expected output includes the post condition and the actual expected output. The test cases have an identity, purpose, date of execution, results, creator, and version information to support the management. Hence, test cases need to be developed, reviewed, used, managed, and saved as shown in Figure 2.8 (Jorgensen, 2008).

TEST Case ID			
Objective			
Pre-Condition			
Input			
Expected Output			
Post-Condition			
Execution History			
Date	Result	Version	Run By

Figure 2-8 Typical Test Case Information adopted (Jorgensen, 2008)

2.4.4 Test Case Management Systems

A Test Case Management System (TCMS) is a system in which test cases can be created, modified, retrieved, restored and traced (Tauhida et al., 2007). The motivation of a TCMS could be to minimize the pain and times spent on auditing and tracking all the test artefacts (Majchrzak, 2010). In addition, a TCMS starts with a test case template or a graphical user interface, which guides the testers to construct a well-structured test case. The number of test cases will approach into the hundreds of thousands or even millions. Microsoft for instance, which will be discussed further in the coming chapters, developed one million test cases to evaluate the Word application.

Desai (1994) developed a TCMS using object-oriented design and relational database to support management of test cases and test results, maintenance of a standardized test case format, execution manual as well as automated test cases and generation of customized reports. In managing test cases, the system provides the storage, retrieve

and updating of test cases using command line and/or user interface. Furthermore, Rex (2002) enriched the literature with his team experience in developing a test management system based on their practices. A recent implementation for a web TCMS showed how the quality and efficiency of testing process improves among its users (Yuan, 2011).

2.4.5 TCMS Attribute

The test management tool includes features to assist on test planning, current test tracking and aiding the traceability. This tool in its basic form contains a standard test case template, an upload feature, test organizer, a historical data retrieval feature, and a summary report of the tests. An additional factor in an advanced tool may include a series of templates in which the end user fills in the fields that structure the test case. Building the relations of the test cases with other testware and artefacts will be very useful features in re-using them.

On the other hand, tracking test cases is a task to allow management of the test process for any mentioned project. Nevertheless, test case management is not just about tracking test cases, but it also involves organizing testing artefacts in a systematic manner (Tauhida et al., 2007). The most vital element of any test case management tool is how it represents the test cases for making them easy to be manipulated by a third party, regardless of their level of testing knowledge.

2.4.6 TCMS Differing Factors

To have an efficient TCMS tool certain factors have to be considered when we develop or choose any one of these tools. These factors make the tools differ from each other in their performance and results. These factors have been identified and discussed by different interest groups from both academic and practitioner based on research and experience such as in (Chunyue, 2011; Damm, Lundberg, & Olsson, 2005; Louridas, 2011; Mordechai, 2008).

Figure 2.9 illustrates the main factors used in a sophisticated TCMS approaches as stated in (Louridas, 2011) as follows:

	Seapine TestTrack	QMeTry	TestRail	XStudio
Interoperability with issue-tracking systems	Yes (with Seapine TestTrack Pro)	Yes, with Mantis, Bugzilla, and JIRA Enterprise Edition	Yes, through URLs	Uses its own bug-tracking database, connectors with JIRA, Trac, Bugzilla, and Manti
Interoperability with authentication/ authorization systems	Yes, it can use Single Sign-on	No	Yes, it can use Single Sign-on	No
Integration with requirements	Yes, via TestTrack RM	Handles requirements internally	Yes, through URLs	Handles requirements internally
Integration with source control tools	Seapine Surround SCM, CVS, Clearcase, PVCS, Perforce, SourceOff-SiteClassic, StarTeam, Subversion, and Visual SourceSafe	No	No	No
Integration with developer tools	VisualStudio and Eclipse	No	No	No
Interface	Client program, Web interface (for subset of functionality), and SOAP SDK	Web interface	Web interface and API for submitting test changes and test results	Client program, Web interface, and SDK for integrating with existing tests
Platforms	MS Windows, Mac OS, and Linux	Available as hosted software as a service; on-premise installation MS Windows and Linux with PHP and MySQL (for instance, XAMPP)	MS Windows server 2003 or 2008, IIS with FastCGI/PHP integration; Unix-based OS with Apache, MySQL, and PHP	Java Runtime Environment 1.6 for the server, MS Windows, Linux, Mac OS fat clients, and Web client requiring Web server (Apache, Tomcat, IIS)
Databases	Internal, MS SQL, Oracle, PostgreSQL, and MySQL	MySQL	MS-SQL and MySQL	MySQL
License	Proprietary	Proprietary	Proprietary	Free; parts are open source

Figure 2-9 Test case management tools VS. Factors adopted (Louridas, 2011)

2.4.7 Lack of Management

There are certain specialized journals and research interest groups, which have written articles concerning quality management in IT development. As a result, there is a significant amount of applicable and important literature in that field. However, although they address many methods and approaches to quality management, practically none of it intended to address the issue of management of IT and software assets. A study in ("Lack of Test Case Management Threatens Software Quality," 26 June 2008) revealed that only approximately a quarter of business organizations are utilizing any TCMS application at this time. In the Figure 2.10 according to the study, it is shown that the percentage amount of manual testing process is quite low compared to an automated TCSM. TCSM is still in its infancy, and therefore the manual process of analyzing of the sheer amount of test cases produced in every testing process makes it impossible to link the individual test cases to their test-ware and effectively utilize them as management assets.

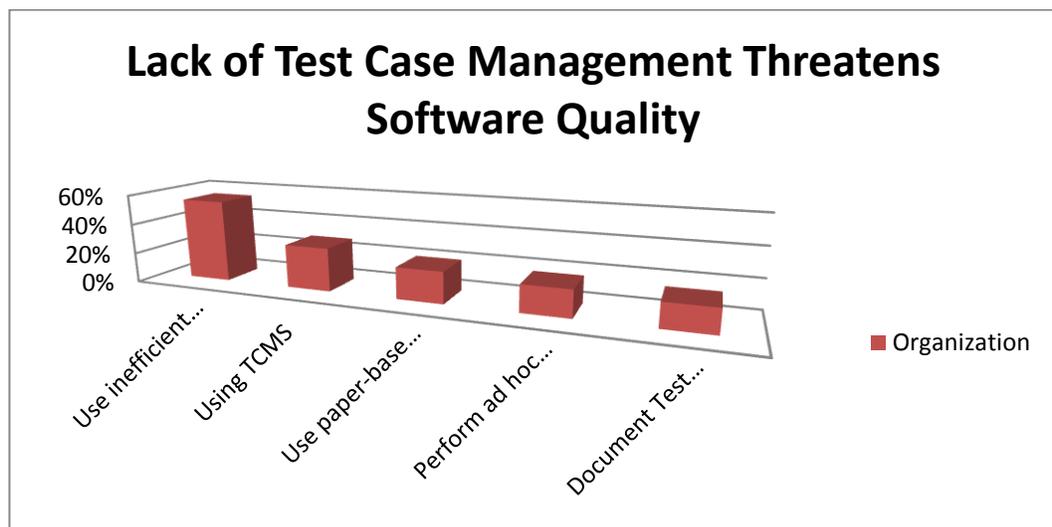


Figure 2-10 Number of Organization using TCMS

Lack of utilizing available TCMSs is not because of any shortage of substantial amount of them, but because of misunderstandings created between the business project and technical management. Software engineers do not really understand business management, although you can find their management applications in almost every business and office nowadays. They only develop the management applications in every field based on their technical knowledge, not from a standpoint of overall business acumen. The software engineers understand management only in the terms of technical configuration management (i.e. versioning) for developing the products, so they rely on the project managers to lead the project. However, at this stage, the project managers need help from the technicians to understand the software. This lack of understanding creates a “disconnect” in communication, which contributes to the costly and delayed end product.

2.5 Summary

Semantic web technology holds various promises for developing efficient conceptual web data represented in a formalism approach, which can be meaningful to be accessed by third parties, regardless if human or machine. It has shown efficient results on search engines, agents, personal desktops, knowledge management and other areas. The *web of data* is structured in several technology layers, which works together to fulfil the required tasks as noted earlier. Since the semantic technology idea opens up many possibilities of harnessing the linked data, there is a high chance that the technology can bring many benefits in developing semantic test case management systems (TCMS).

Moreover, in this chapter we overviewed basic concepts, technologies, and applications of semantic web technology and ontology-base systems. Furthermore, with the help of the bookstore data example, we illustrated how the semantic technology differs from the current syntax technology. Finally, we provided examples of applications of semantic technology and ontology-based systems in the knowledge management area; and from that, we foresee that semantic web technology and ontology-base systems stand out as a promising technology for knowledge development and management.

As aforementioned, a test case underlies the effectiveness and efficiency of the testing process. Each test case comprises management information, such as IDs, creator, date, and version; condition information, such as prefix and postfix values; and the input and expected output information refer to our discussion in section 2.2. It was pointed that a TCMS is to be featured with create, store, retrieve, and update for test cases, to contain a standard test case template and to be able to summarize the test results.

Nevertheless, as long as the process remains manual, testers continue to battle the challenges of the lack of management. However, the amount of management systems available, as shown in section 2.3.2, is not providing any usable solution; especially in relation to test case reusability refer to section 2.3.3. Although journals and special interest groups for testing management exist, the lack of utilizing efficient management in testing is an issue.

The related problems are discussed further in chapter 3. From this, we argue the following:

- I. The loss of definition to test terms renders initial test cases un-reusable. When automation is implemented, information can be extracted with precision regardless of the term chosen for the search.
- II. Incorporating Software Testing Ontology to support wider use of the automated process allows for various words to be defined as synonymous terms.
- III. By integrating semantic search technology in TCMS, the search will become a more productive experience for testers.
- IV. Well-structured individual test cases are applicable to be represented in Semantic Test Case Management (STCM) system.

3.0 Limitation of Test Case Management

To support wider use of Test Case Management System, it is important to capture and represent not only just test cases, but also other related testware and software artefacts. This is especially useful in supporting analysis tasks for the purpose of reusing test cases, where association of metadata into test case structure as well as provision of vocabulary in the form of ontologies is required. This chapter addresses the problems in the software testing process in the following sections:

3.1 Automation

Test automation has a collection of promises. These promises might include efficient performance, run more tests than manual, perform tests that could be unreachable by manual test, and reuse of test. Using automated test tools would facilitate the testing process.

For instance unit testing, even for small program would need a huge manual task and as the program grows, that task would be overwhelming. Likewise, automating the software testing process would not only facilitate the process, but it would also minimize the human error and extensively reduce the total cost as testing costs up to 50% or more in a safety critical applications of the software development life cycle (Ammann & Offutt, 2008).

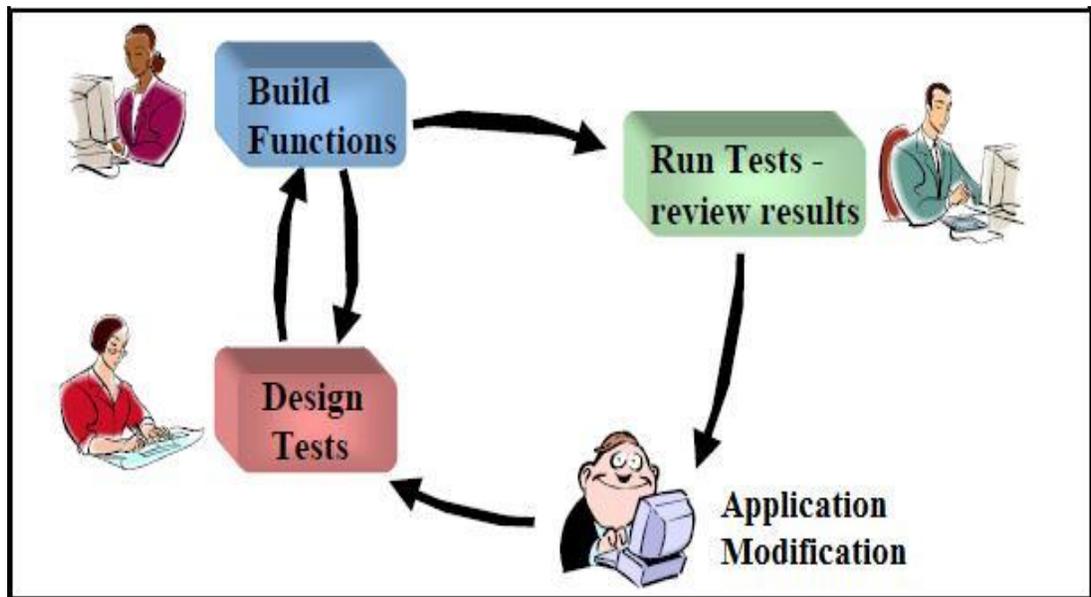


Figure 3-1 Automated Tools Model

In figure 3.1, we show the assumed simple automated process models representing the manual process. Automated tools and systems supporting software testing process become a key technology for today's software industry. However, complete deployed 100% software testing automation system is a goal for the long term of research. The research towards that goal is active in either test generation or innovative support procedures.

Furthermore, one way of minimizing cost and maximizing efficiency is to reuse the test cases (Bertolino, 2007). However, Hayes (2000) said "A *test automation system is an application which allows a test case written in a tester-friendly format to be executed and the results reported*". In addition, Yahaya (2008) pointed out, the development of knowledge-based software testing system is more achievable with the advantage of semantic technology concept.

Therefore, here we proposed an ontology-based semantic test case management system as it gains its advantages from the power of semantic technology to provide automated support for creating, modifying, retrieving, restoring and tracing individual test cases that are linked with other testware and artefacts.

3.2 Individual Test Case

As stated above, the research towards comprehensive fully automated testing is active. In this case we consider the importance of test management. From that, we focus on the problem of managing individual test cases. Obviously testing systems as a daily task would require writing and executing tens of thousands of test cases, which in turn, will lead to correcting thousands of errors found, handling their links to test artefacts such as modules, documents, codes, etc. Hence, management is a must (Louridas, 2011; Myers, 2004). Moreover, if testers do not keep track of the test cases to be run, how can they gauge the test coverage later on? This question been highlighted by (Rex, 2002). However, taking into account that test cases can be thousands and millions in numbers and without a way organizing, storing, and retrieving them, it could be a pool of mess (Patton, 2001).

The current test case management systems use keywords matching as a search method combined with information retrieval rather than semantic search method combined with conceptual information retrieval as we going to see in the coming sub-chapter. In addition, these systems offer limited representation of the linkages between test cases and related testware and software artefacts as well as they don't support the individual test case reusable information.

In this context, we address the aforementioned weaknesses by first finding a good way of representing the individual test case. We focus our research work on the individual test cases themselves, in line with most computer science researchers that have already continuously been doing so by focusing on test suites, as mentioned in (Miller & Voas, 2006). Most researchers have focused on working in the test suites by creating methods for evaluation and automatically generating test suites, and they pay less attention to the individual test cases. The power of individual test cases helps to identify bugs in the test process, for instance in mutation testing of System X (hereafter called *the System*). There is software, which will create different versions from the System, so we end up with Systems X1-Xn, which are only slight differences from the original. The individual test case power will be clear during this part of testing, where we create different individual test cases, which creates a base of results for the System. After testing X1-Xn, we will compare the outputs with the original results to confirm the absence of bugs in the System. The individual test cases should be utilized for the purpose of reusability.

Reusing test cases, instead of creating new ones in each instance, will prove the usability of this process as reusing software components has proved to be cost efficient for software products. There are several problems, however, with being able to reuse these components. The main problem that concerns our research in software component reuse is how to find the component that should be selected for this process. The same problem may occur and arise whenever reusing individual test cases is desired. Furthermore, we found (Nakagawa, Simao, Ferrari, & Maldonado, 2007) shed some light on this situation that testing-tools developers lack consideration, paying less

attention to the evaluation, maintenance, and reuse features on the tools design than saving the cases for reference and reporting.

We say the solution to this problem is by representing the individual test case semantically, which means linking them to the testware and other test artefacts. So in this scenario, presenting these components (individual test cases) in a conceptual framework would definitely help the end-users to search for them in an efficient way. By doing this, we will produce a better representation for test case management. This representation is being utilized in our ontology-based test case management system. At first, we identified the well-structured items of the individual test case based on the literature review, and then represent each item in the Resource Description Framework. The whole process is described in the implementation chapter.

3.3 Software Testing Terms

Testing terminology comprises of all terms that belong to the testing process in the software engineering domain of knowledge. Defining standard terms in any domain will benefit all parties working within that specific field. Machines, understanding how to manipulate the process, will also gain the same benefits gained by humans from standardizing the terms in that field.

With testing terminology, different personnel involved in the testing process might use different terms for the same item. For example, one test manager calls an item a “bug”, while the tester calls it “error,” and the programmer calls it a “fault.” All three personnel are actually referring to the same item; however, by using different

terminology, they may think the other is referring to a separate item. In another scenario, if this data is input into a machine for an automated testing process, the machine cannot detect that these three terms are actually synonyms referring to the same item, not three separate things. This causes delay in the job or work due to the confusion caused. We found that, in a case study that was held in an industrial setting, it was found that the job was delayed due to the terminology confusion (Tauhida et al., 2007).

We say that a shared understanding among different terminology would definitely; overcome the overlapping and miss-matched concepts, benefit the knowledge integration, and potential the re-use of sources. Computer science is one of the fields that has benefited from ontology mechanism. This mechanism is simplified as, modelling the domain knowledge constructed with the reasoning behaviours. This mechanism enables the end users to reuse knowledge. Knowledge reuse is a higher level practices allowing the share of common terms and concepts of a domain.

3.4 Search Technology

In TCMS tools, it is crucially important to have the technical capability to be effective in its usage. If we refer to the TCMS in general, where millions of test cases are stored therein, using the traditional search algorithm would end up with an enormous volume of isolated test cases again. Obviously, these results would not be effective for the testers to actually reuse. It would be easier for them to create new test cases instead of searching in this manner for one to reuse (Fraser & Zeller, 2011).

Search is a feature that allows end users to search and retrieve documents. Essentially, search programs have different approaches, for example a search based on a combination of textual keywords with an importance ranking of the documents. This traditional search algorithm has various limitations, which focus on the frequency of the word appearance in the documents. Most of management systems use the traditional approach to searching for the assets they are managing (Tonta, 2011). This information lacks a semantic approach to the searching results (Juan, Lizhi, Weiqing, Zhenyu, & Ying, 2009). There are large numbers of Testing Management Systems. Table 3.1 identifies the matrices followed on selecting the required systems. Our selection was based on the: web-based application, open source, method of the domain search and technique of storing the data. In Table 3.1, we can observe that testing management systems do not emphasize on the search feature and storing mechanism. As a result, the reuse facility is not considered as a factor, which our system overcomes.

Table 3-1 List of Sample Test Management System

System	Application Base	Search	Storing
Approach			
ApTest Manager	<i>Web based Test Management</i>	<i>Not applicable</i>	<i>Keyword Index (Database)</i>
Chrysilla Test Case	<i>Web-based service</i>	<i>Not applicable</i>	<i>Keyword Index (Database)</i>
TestUP	<i>Web based Test Management</i>	<i>Keyword Search</i>	<i>Keyword Index (Database)</i>

One of the current trends in searching is utilizing semantic search approaches. It is an improved form of search, where meaning and structure are extracted from the user's search

queries to be exploited during the search process. One of its capabilities is to provide more information in the search results (Hendler, 2010). For instance, in TCMS the results of the semantic search will not deliver an isolated test case, but it will retrieve other related information linked to the targeted test case. This capability will facilitate the tester by greatly improving the chances of finding the right test case to be reused. Not only that, but it will provide the user with other relevant information to help them in expanding their search scope in a related way. For illustration, if a tester is searching for a test case with a specific test environment, the engine will provide the tester with not only the specific test case but also with all test cases that have been executed in the same environment. This scenario will significantly aid the tester to find more test cases related to this same attribute. The semantic search approach will change the search experience in the testing domain knowledge. Therefore, we utilize the semantic search approach in our Semantic Test Case Management System to increase the degree of test case reusability.

3.5 Summary

It's time to restate that this chapter's intent is to view the weaknesses regarding automating and managing test assets, in particular test cases. So, we have covered the concepts of automation, individual test case, testing terms and searching algorithm. To show the significance in making the testing process manageable, we proposed four main aspects: (1) automated information extraction (2) representation of well-structured test cases (3) incorporation of software testing ontology, and (4) semantic searches. In this direction we consider designing the four aspects to prove the strength of what we call Semantic Test Case Management System in the following chapter.

4.0 Ontology-based Semantic Test Case Management

The work presented here is novel in the following:-

- I. its automated information extraction,
- II. its application of well-structured representation of individual test cases for the Semantic Test Case Management (STCM) system,
- III. its incorporation of Software Testing Ontology to support wider use of the STCM system and
- IV. its integrated semantic search technology

4.1 Automated Software Testing Information Extraction

Semantic Web is considered as the future web and it promises to unburden users to retrieve heterogeneous information as it will have well-defined meaning. To understand how it works, we are outlining the collection of the main concepts and standard technologies and providing an example of bookstore data that need to be published and processed by a third party. They are organized, in a form of, illustrating how the data is identified, represented and accessed in the web.

- **Data Identification**

The data represented in the semantic web have a well-defined meaning and contain information about their contents in the form of metadata. Metadata is defined as *“Metadata is data about data. The term refers to any data used to aid the identification, description and location of networked electronic resources. Many different metadata formats exist, some quite simple in their description, others quite*

complex and rich” (IFLA, October 24, 2005). To narrow down the Metadata definition to the content of web Berners-Lee (January 6, 1997) define it as “*Metadata is machine understandable information about web resources or other things*”. The use of metadata is to represent a shared understanding data which can be processed by not only human but also machine (Antoniou & Harmelen, 2008).

In our example, the info of the test case can be represented in a standard metadata such as Dublin Core (DC) that was proposed in a workshop held at Dublin, Ohio 1994. The simplicity of DC elements that describes resources was behind its popularity (Core, 1995). Hence, we select the elements (Subject, Creator, Source, Purpose, and Identifier) from the DC elements to represent the main basic data of our TestCaeDetails example as demonstrated in Table 4.1.

Table 4-1 Representing TestCaeDetails Data

Subject	Creator	Source	Purpose	Identifier	RDF
Login	Mansoor	University Malaya	Validate User	TC-ID = 0001	www.um.edu.my

This metadata can be presented in XML syntax as follows:-

```

<TestCaseDetails>
  <TestCaseID =TC-ID >
  <Subject>Login</Subject>
  <Creator>Mansoor</Creator>
  <Source>University Malaya</Source>
  <Purpose>Validate User</Purpose>
  <RDF>www.um.edu.my</RDF>
</TestCaseDetails>

```

- **Data Representation:**

The Web of Data is providing languages that allow the combination of data and its rules of reasoning to be represented. The official recommended by World Wide Web Consortium (W3C) languages such as RDF-RDF(S), and SPARQL built up to serve various purposes of managing the knowledge at the web (Grobe, 2009).

- **RDF-RDF(S):**

RDF basic concept is to represent the metadata which describe the resources and using URIs to identify them. Meanwhile, RDF(S) basic concept is to allow the users to identify their own vocabulary schemas: e.g. class hierarchy (Gupta, Malik, Prakash, Rizvi, & Arora, 2004). The following Figure 4.1 is to illustrate RDF triples (object-attribute-property) that forms RDF statement (subject, predicate and object) using our bookstore example.

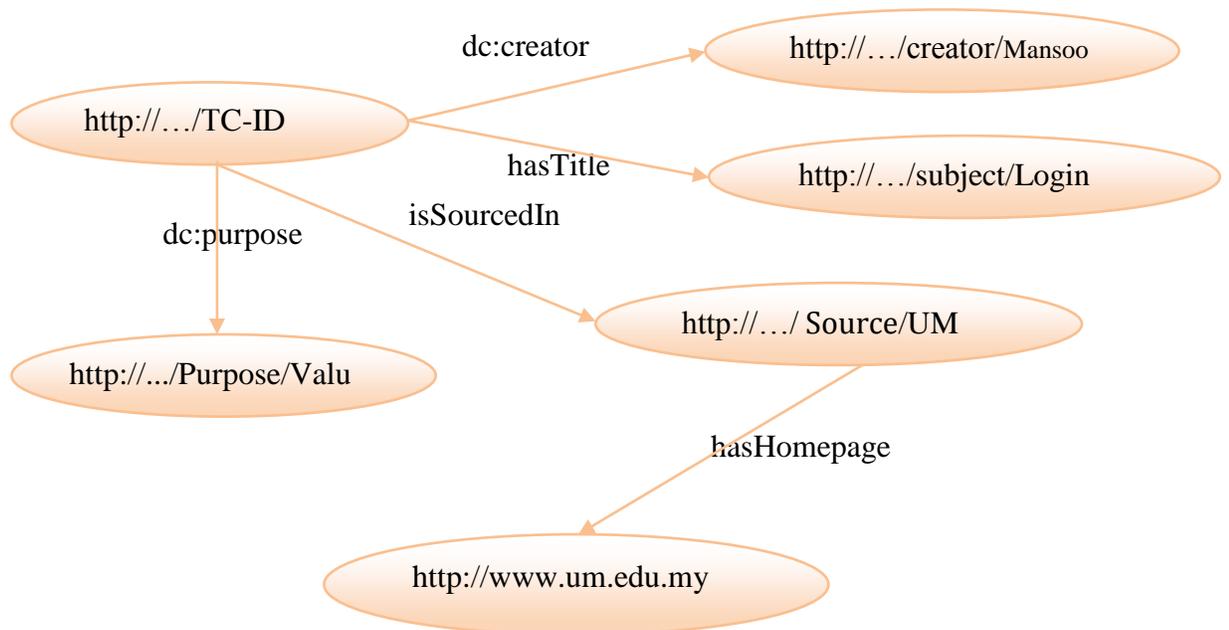


Figure 4-1 Graph Representation of RDF Triple

- **SPARQL:**

It is considered as the query languages for the semantic web and gained the W3C recommendation, the essential idea is to use graph pattern to query the RDF data and retrieve the data in a form of a result set or RDF graph contains the existing recourse references and their relations (Antoniou & Harmelen, 2008).

- ✓ For instance we are using the SPARQL basic example to query the ISPN in our Bookstore Data.

```
SELECT?TC-ID
WHERE
{
  http://www.um.edu.my/Identifier?TC-ID
}
```

- ✓ The results of this query will be as follows:-

TC-ID
0001

- **Data Logic Representation**

The underlining aim of semantic technology is not just to present data on the web but to provide the data with a facility to reason the knowledge that need to be represented. Hence, one of the main components of the web of data is OWL as it defines the reasonable subset of logic (Antoniou & Harmelen, 2008). Logic is considered as the foundation of knowledge representation as it was described in (Brachman & Levesque, 2004).

- **OWL:**

OWL fundamental model is to present the resources semantically for machine process. Semantically would be achieved by describing the meaning of the resources and supporting its reasoning. OWL is defined in three different sub languages: OWL Full, which uses all primitive of the language; OWL DL (**Description Logic**), which supports the logical reasoning; and OWL Lite, which uses simple restriction (Antoniou & Harmelen, 2009).

- **Description Logic (DL):**

DL underlining aim is to give a logical formalism to model the class, property and instance of an application domain. Concept means here, the set of individuals which the domain data representing, the role is the binary relation among these concepts and the instances are representing the individuals (Breitman et al., 2007). For instance and expressivity on representing our bookstore data we show the construction of DL to describe some data of our example in Table 4.2:-

Table 4-2 Representing TestCaseDetails Data in Logical Formalism

Concept	Role	Contents
Subject	hasSubject	“Login”
Creator	isWrittenBy	“Mansoor”
Source	isSourcedIn	“University Malaya”
Purpose	hasPurpose	Validate user
Identifier	hasValue	“0001”
Homepage	hasValue	“http://www.um.edu.my”
Construction		Comments
0001 ⊆ TestCaseDetails		The Identifier book is a subclass of TestCaseDetails
“Login” ∀ hasValue 0001		The value is use a universal restriction only as a unique ID for the title of the test case
0001 ∃ isSourcedIn “University Malaya”		The TC-ID 0001 exist in the publisher UM

4.2 Representing well-structured Individual Test Case

We define well-structured test case as a set of identified variables and terms that are linked to serve the semantic of the related software artefact and software testing terms required for the third party (human or machine) to understand, and to complete the reasoning of the test cases.

4.2.1 Design of Well Structured Test Case:

Standard test cases would include attributes and metadata. We refer the attributes to the information or values that are for conducting the test and the metadata to the information or values that are required either for identification, selection, discovery or other analyses on test cases, and they are not directly relevant to the act of conducting the required test itself.

I. Attributes as shown in Figure 4.2 and described in Table 4.3:

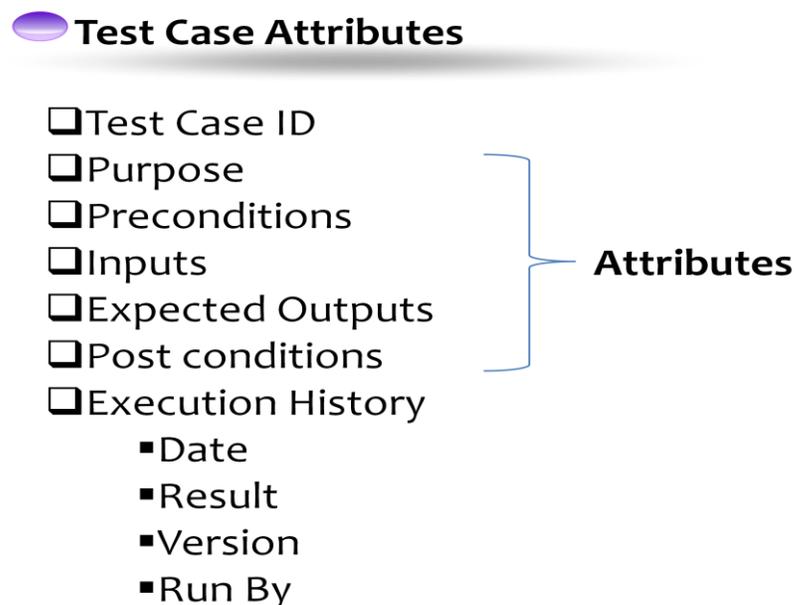


Figure 4-2 Attributes of Test Case

Table 4-3 Brief description of the Test Case Attributes

Attributes	:	Description
Purpose	:	Data describes the objective of the Test Case
Precondition	:	Data describes the conditions that must be met before the Test Case is executed
Inputs	:	Data describes the steps of the Test Process
Expected Outputs	:	Data describes the results of the steps of the Test Process
Post conditions	:	Data describes the conditions that must be met after the Test Case is executed

II. Metadata as shown in Figure 4.3 and described in Table 4.4

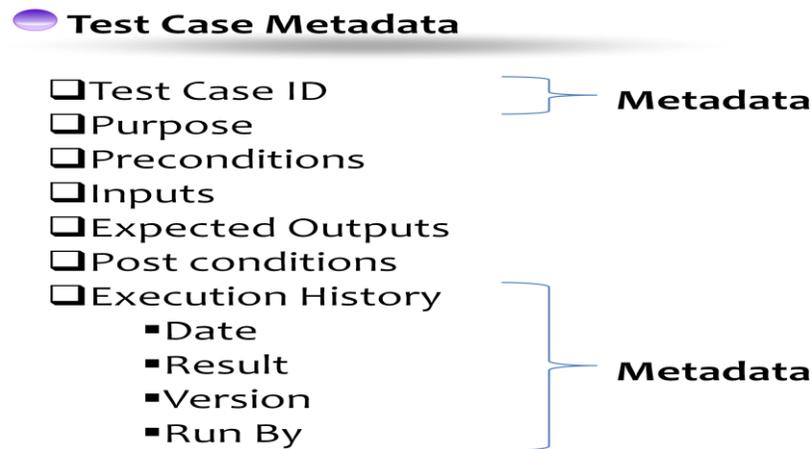


Figure 4-3 Metadata of Test Case

Table 4-4 Brief description of the Test Case Metadata

Metadata	:	Description
ID	:	Data to present the identification of the Test Case
Result	:	Data to present the status of the Test Case
Version	:	Data to distinguish between different revisions of the Test Case
Run By	:	Data of the Test Case creator
Date	:	Data of the date captures by the system

4.2.2 Design RDFS for Test Case:

Semantic technology allows users to build their own hierarchy schema (RDFS), and by utilising two metadata schemas that are connected to semantic initiative

1. Dublin Core (DC) (Weibel, Kunze, Lagoze, & Wolf, 1998) and Test Metadata
2. W3C Quality Assurance Work Group (QA) (W3C)

We designed our own hierarchy schema to represent individual test case semantically as illustrated in the following steps:

- I. **DC:** defines 15 elements applicable to resources in general as shown in Table 4.5.

Table 4-5 Dublin Core Elements Set

<i>Contributor</i>	<i>Format</i>	<i>Identifier</i>	<i>Relation</i>	<i>Subject</i>
<i>Coverage</i>	<i>Date</i>	<i>Language</i>	<i>Rights</i>	<i>Title</i>
<i>Creator</i>	<i>Description</i>	<i>Publisher</i>	<i>Source</i>	<i>Type</i>

- II. **QA:** claims the minimal set that can be applied to test case as shown in Table 4.6.

Table 4-6 Quality Assurance Elements Set

<i>Identifier</i>	<i>Title</i>	<i>Purpose</i>	<i>Description</i>	<i>Status</i>
<i>SpecRef</i>	<i>Preconditions</i>	<i>Inputs</i>	<i>ExpectedResults</i>	<i>Version</i>
<i>Contributor</i>	<i>Rights</i>	<i>Grouping</i>	<i>seeAlso</i>	

III. Common Elements:

It is observed that there are several common metadata terms for both the schemes as shown in Table 4.7.

Table 4-7 Common Elements Set

Elements
<i>Identifier</i>
<i>Title</i>
<i>Contributor</i>
<i>Description</i>
<i>Rights</i>

IV. Principles:

To derive the RDFS of test case structure, we followed the following principles:

- ✓ Use the prefix dc for the terms borrowed from DC
- ✓ Use the prefix qa for the terms borrowed from QA
- ✓ Use the prefix xx for new defined terms
- ✓ Obliterate non-required terms

V. Well-Structured Test Case:

Well structured test case should identify terms that are related and serve the concepts of the **STO** as it facilitates the semantic of the testing terms required for the third party (human or machine) to understand and to complete reasoning on test cases. The mapping is demonstrated in Table 4.8.

Table 4-8 Mapping Test Case Terms to STO Concepts

Prefix	Term	Concepts
Rdf	about	Artefact
Dc	Creator	Tester
Qa	Purpose	Artefact
Dc	Source	Artefact
Dc	Subject	Artefact
Dc	Relation	Artefact
Qa	Preconditions	Environment
Xx	TestType	Task_Testing
Qa	Input	Artefact
Qa	Expected Result	Artefact

From Table 4.8, can observe that we have done the following:

- ✓ Represented the Metadata Test ID with the rdf: about as it is provided by the rdf scheme.
- ✓ Replaced the Metadata Run By with the dc:creator as DC is well established.
- ✓ Utilised the DC source, subject and relation terms to represent the test case for the following reasons:-
 - *dc:source* is useful for test case analysis for re-use purposes as it links the test case with its artefact.
 - *dc:subject* is required to interpret the test case for searching purposes.
 - *dc:relation* is a technique element to show the relation of the test case with other test cases.
- ✓ Included a new term xx:TestType to relate the test case to the Software Testing process tasks.

- ✓ Obliterated the Metadata date, result & version as date can be captured by the system whenever the test case is created; version when the test case is edited; and result is not required to be represented.
- ✓ Obliterated the attribute Postcondition as it can be included with the test case Purpose and not required as a separate term.

VI. Test Case RDFS:

We represented the Well-Structure Test Case terms in RDFS schema as shown in Figure 4.4. Note that **xx** is implemented as the default name space.

```

1  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="
   http://www.SWbSTS.edu.my/" xmlns:qa="
   http://www.w3.org/TR/2005/NOTE-test-metadata-20050914/">
2  <rdf:Description rdf:about="http://purl.org/dc/elements/1.1/#TC1">
3      <qa:Purpose>Purpose</qa:Purpose>
4      <dc:source> Source</dc:source>
5      <dc:subject>Subject</dc:subject>
6      <dc:relation>Relation</dc:relation>
7      <dc:description>This resource contains Test Case</dc:description>
8      <dc:creator>Creator</dc:creator>
9      <TestType>Type</TestType>
10     <qa:precondition>Precondition</qa:precondition>
11     <qa:input0>Input 0</qa:input0>
12     <qa:ExpectedResult0>Expetected Result 0</qa:ExpectedResult0>
13     <qa:input1>Input 1</qa:input1>
14     <qa:ExpectedResult1>Expetected Result 1</qa:ExpectedResult1>
15 </rdf:Description>
16 </rdf:RDF>
17

```

Text RDF/OWL
TestCaseRDF

Figure 4-4 The Well-Structure Test Case RDFS

VII. Automat Software Testing Information Extraction Component Design

The design is based on jOWL (jQuery plug-in for navigating and visualising OWL_RDF) for the Software Testing information extraction, which provides a set of gates to browse and navigate between different terms, taxonomy and relations in the STO.

4.3 Incorporation of Software Testing Ontology

An ontology-based system is about featuring intelligence and insight provision capabilities. In order to deliver the relevant knowledge, the ontology-based system takes into consideration the ontology changes according the business environment. Ontology-based systems attract software testing researchers where we can find numerous attempts to solve various software testing research problems using ontology-based systems. For instance an ontology-based question answering system on software test document domain (Serhatli & Alpaslan, 2009) is concerned with retrieving test documents and discusses how to improve the searching by ‘questions and answers’ in a natural language. The work proposed a new algorithm to filter the question tokens and asserted to the reasoner to retrieve exact information. This work been implemented to retrieve related testing documents for new member joining the testing team. Another example is an ontology-based approach for GUI testing (Han et al., 2009) which is concerned with generating GUI test cases according to existing testers’ experience and information provided in the ontology. The ontology stores information about the GUI and makes use of the reverse engineering to capture the knowledge provided.

We conclude that ontology-based systems is an established discipline as shown in the empirical study by (Simperl et al., 2010). Moreover, ontology-based systems provide a shared knowledge model and become an important area to solve the problems in knowledge management systems. (Fu, Yue, Song, & Xin, 2008).

In figure 4.5, we demonstrate the term Error in ontology showing how it is related to the other similar terms that belongs to the same concept.

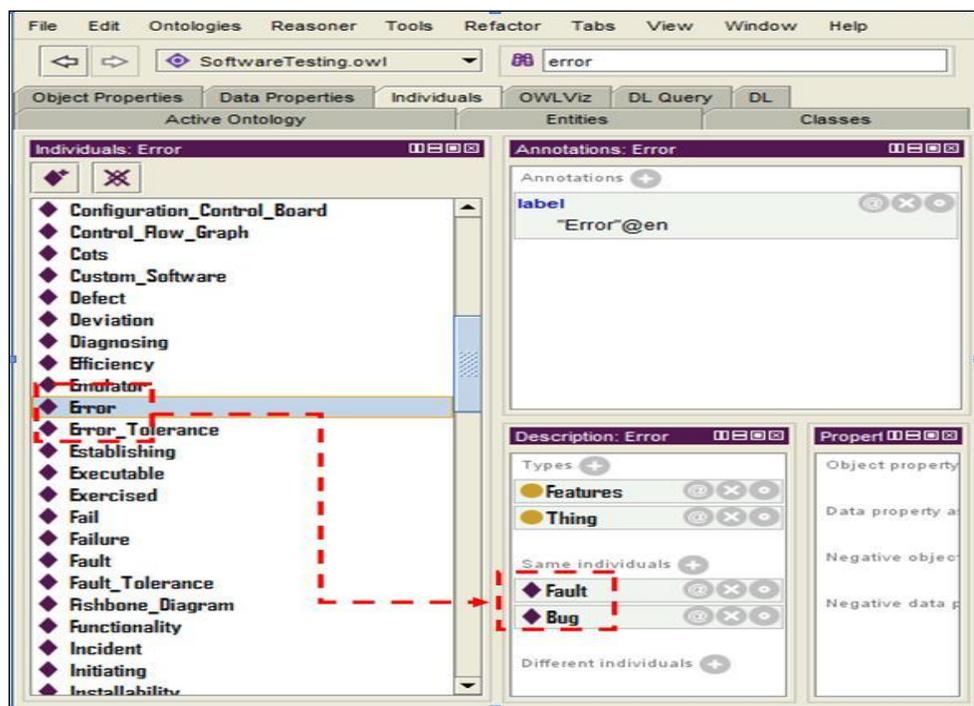


Figure 4-5 Demo the Term Error with similar Terms

Furthermore, ontologies are main components in Semantic Web. They represent the domain knowledge enabling the cooperation between people and machine to machine. Building up an ontology will help the testers and other involved personnel, whether machines or humans, in reusing these terms efficiently.

In conclusion, standardizing the terms and putting them in a conceptual manner will minimize the confusion in the testing process, resulting in more efficient software production. When the confusion is eliminated, the testing time is reduced, which usually takes 50% of the software life cycle production. This time reduction will affect the total cost of building software, producing a more time and cost effective testing process.

We have built conceptual terminology into software-testing ontology [STO] covering a standard testing glossary. Our STO consists of hundreds of concepts of software testing based on the mentioned standard testing glossary, which is large enough to supply accurate reasoning terms for our STCMS. It also defines the relations between these concepts, for instance as “isTestedBy” and “hasTestID.” This is elaborated on in the STO implementation chapter.

4.4 Integration of Semantic Search Technology

The semantic search will deliver test cases and retrieve other related information linked to the targeted test case. This facilitates the tester by greatly improving the chances of finding the right test case to be reused. It provides the user with other relevant information to help them in expanding their search scope in a related way. To achieve the approach we discuss the design in the following:

- I. The purpose of interpretation,** we illustrate the Login Test Case example to show how the data are created and represented based on RDFS.

II. Logic Innovative Service: we present the innovative services provided by illustrating the authentication (Log-in) Test Case example. Table 4.9 presents the test case description.

Table 4-9 Login Test Case description

System: MFIT FoodReg		
Test Case No.	1-MFIT_Login	Test Case Version Version 1.0
Test Title	Login	
Test Objective	To validate the entered User name and Password	
Pre-requisite	Valid username and password	
Tester	Mansoor	
Step	Description	Expected Result
1	Fill Username field	Username is entered
2	Fill Password field	Password is entered
3	Click Login button	If details are valid login is successful and user page is displayed. Else login is failed and “Alert message is generated”

This test case description is represented in RDFS and saved in the database for future search and retrieve. The logic of the service is shown in figure 4.6. For instance, the test case title is represented logically with (Test Case \exists hasTitle MFIT_Login), creator is a sub class of Tester, and Test Type is an element of Task Testing, which requires interpreting from the software testing ontology STO.

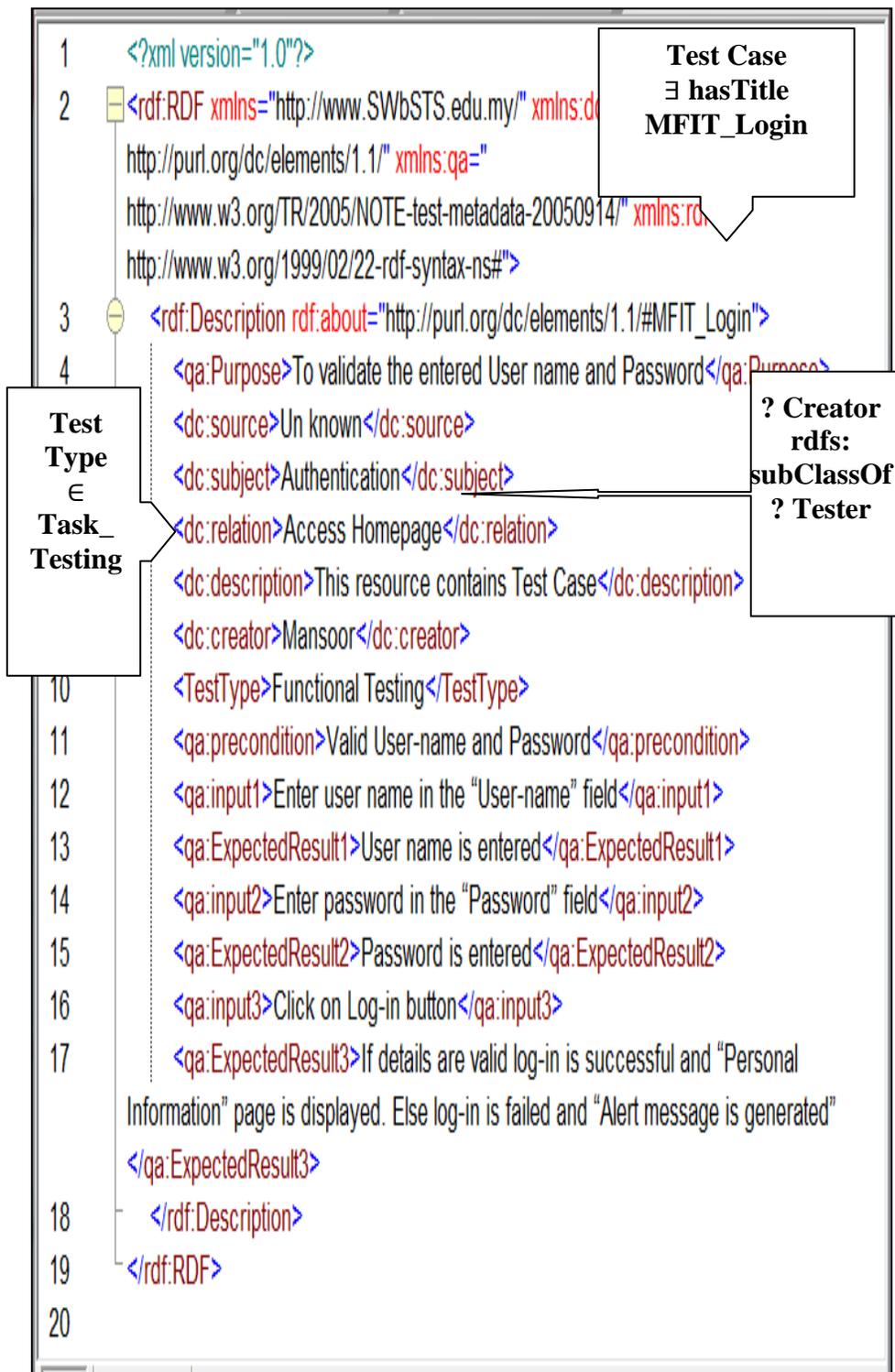


Figure 4-6 Logic Innovative Services of Test Case

5.0 Designing of Software Testing Ontology

Software Testing Ontology (STO) is a formal and an explicit description of software testing concept. STO is about modelling the Software Testing domain. It allows automatic reasoning to provide a formal semantic for the software testing terms (Veenendaal, 2010). STO was built to make software testing domain amenable to be interpreted and processed by third party (i.e. human, machine, software agent, etc). It is considered a semantic repository which manages the storage and query, offers easier integration and dynamic interpretation of software testing data. The semantic repository approach allows easier changes and automated interpretation of the data compared to approach in relational DBMS (John Davies, Rudi Studer, & Warren, 2006). The Software Testing Ontology is presented in the Active Ontology Tab of Protégé as shown in Figure 5.1.

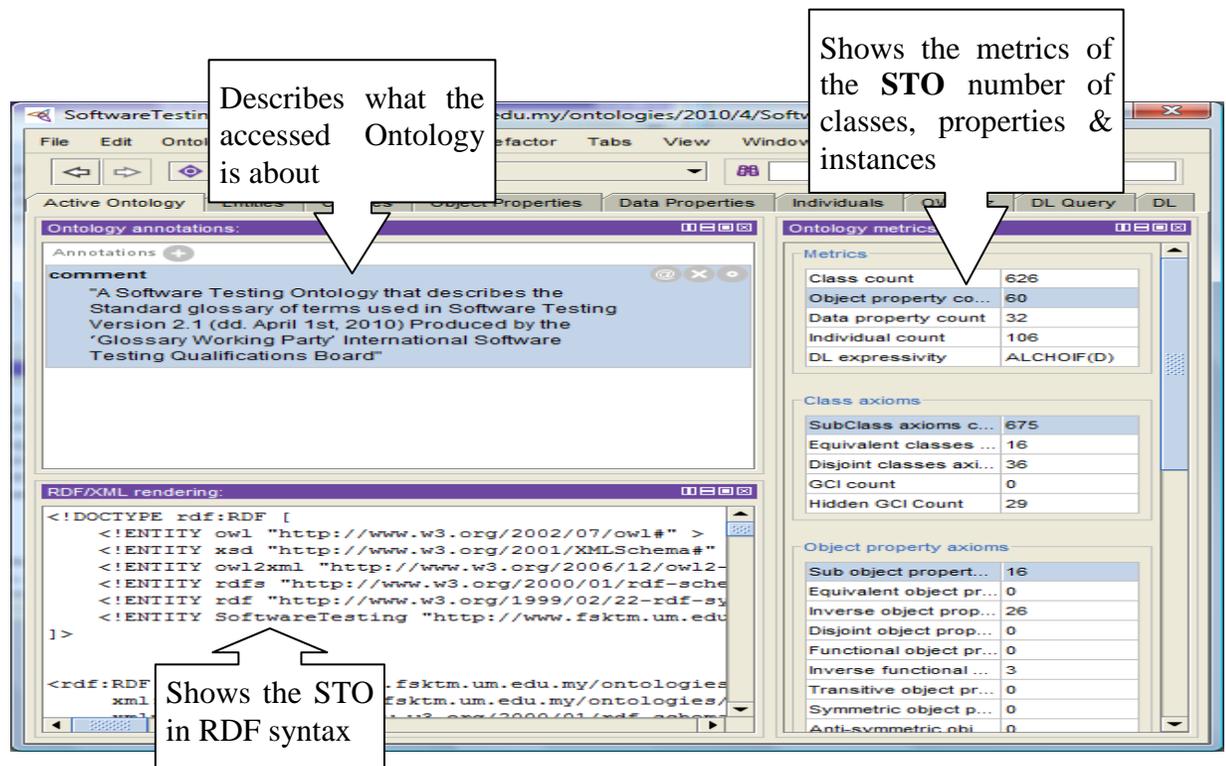


Figure 5-1 STO Active Ontology Tab

For instance, let's store a software testing datum such as **Test Case Suite** that relates to the **Tester** who performs it in a typical relational database. It will be achievable to query the stored data and retrieve typical data such as the **Tester's** details and the **Test Case Suite** that had been performed. Meanwhile, storing the same data in the STO semantic repository as shown in Figure 5.2 enables us to do much more than retrieving typical data.

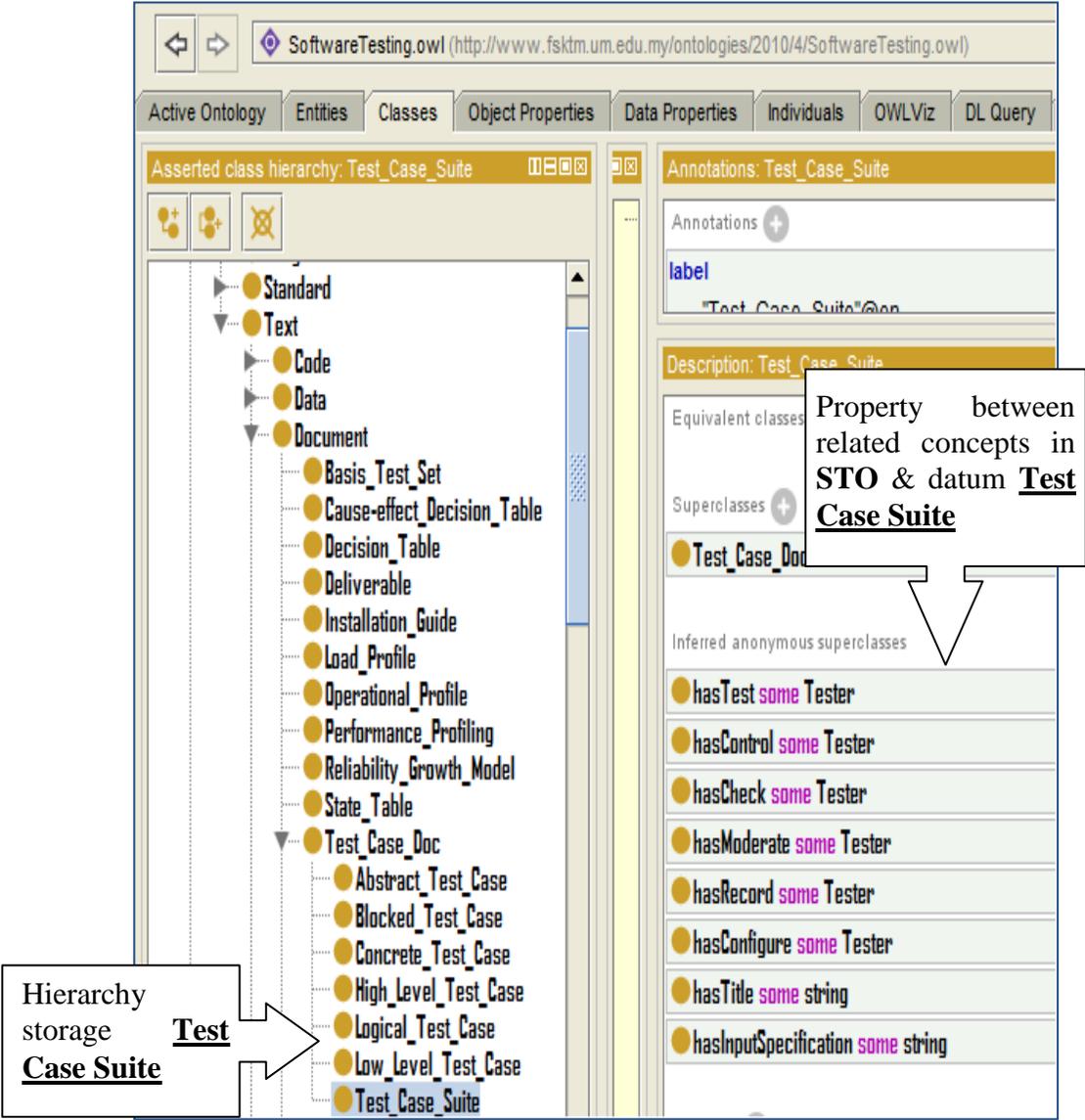


Figure 5-2 Hierarchy Storage Test Case Suite in STO

From the semantic repository in the *STO*, it is possible to deduce more details about the datum **Test Case Suite**: for example, the ability to figure it is a type of Test Case document in a relation property called “**is-a**” which shows that it is a sub concept of the general concept “**Test Case Document**”. In addition to this information, inverse relation from “**hasTest**”, the “**isTestBy**” information will be deduced as it was identified by the semantic repository.

5.1 Building *STO* with the 101 Guide

Building ontologies requires the selection of a comprehensive guide. The *STO* layer creates the logical relationships between test cases and other relevant testware and software artefacts in the software testing domain. Keeping in mind that usually ontologies can be reused, and so does *STO*, we built up the ontology structure in a very high level conceptualization to be more flexible, maintainable and understandable. Furthermore, the Natasha and Deborah (2001) method was selected based on famous approach, simple explanation on how to develop and evaluate the first ontology and clear identified steps. Several challenges were met while building the ontology. In particular, the main one was classifying terms to formalize the conceptualization. This task consumed a lot of effort and required critical decisions. In the aforementioned guide, there are seven main steps to build *STO*, and the following subsections explain the journey that we went through:

I. Determine the domain and scope of STO

To define the domain and scope of STO, we had to answer several selected questions, which had been suggested by the guide to help determine the following goals: purpose, usage, type of information, and who will need the STO. Table 5.1 illustrates the questions & answers used:-

Table 5-1 Questions & Answers determine STO's domain & scope

Question	Answer
What is the domain the Ontology will cover?	The purpose of building this ontology is to cover the software testing area as a Domain & we call it STO .
What is STO going to be used for?	STO is built to be used as an infrastructure for Semantic Technology regardless of which application uses it with the intention of focusing on representing test cases for management and reusability.
What type of answers should STO provide?	STO needs to provide an understandable, conceptualized and linked vocabulary required by the Software Testing Domain.
Who will use STO?	The STO end users are identified as third party whether they are machines such as (Semantic Agents, Semantic Desktop, etc) or humans such as (Software Testers, Test Managers Test Case Creators, etc)

II. Consider reusing existing Software Testing Ontologies

There are Software Testing Ontologies which have already been built and published in the literature. Studying some of the existing ontologies was an important process for this step. Hence Table 5.2 demonstrates the analysed findings of the study:-

Table 5-2 Analysed Findings for Existing STO

Ontology Name	Description	Reference
Ontology of Software Testing OntoTest	Defines software testing concepts in a layered approach. The main layer covers main testing concepts and relations. The sub layers cover Testing Processes, Testing Phases, Testing Artefact, Testing Steps, Testing Procedures and Testing Resources.	(Barbosa, Nakagawa, & Maldonado, 2006)
Software Testing Ontology for WS (STOWS)	Defines concepts related to software testing into two groups: the basic concepts include context, activity, method, artefact, and environment; and compound concepts include tester, capability and test task.	(Hong, 2006)
Test Ontology Model (TOM)	Defined to specify the test concepts, relationships and semantics from two aspects: Test Design such as test data, test behaviour and Test Cases; and Test Execution such as test plan, schedule and configuration.	(Bai, Lee, Tsai, & Chen, 2008)

From the analysed findings table above, we found limitations on the domain terms (especially those related to test case as individual); and relations between concepts and specific tasks. Therefore, instead of reusing the whole ontology, we used some of the concepts' names and built up the remaining concepts on our own to overcome the aforementioned limitations.

III. Enumerate important terms in the ontology

International Software Testing Qualifications Board is a not-for-profit association founded in Edinburgh in November 2002. One of their missions is to promote common language for testers globally. They form groups in different areas of Software Testing and one of the

groups is the **ISTQB** Glossary working group. This group aims to deliver a glossary of testing and related terms (ISTQB, 2002). There are various versions of the glossary as the group keeps updating the new terms when necessary. **STO** was built based on ISQB-glossary Version 2.1 (Veenendaal, 2010) as it presents the most current concepts, terms and definitions of Software Testing domain and the related artefact. All terms and concepts presented in the glossary were covered and the taxonomy was based on our understanding of the domain. In the following section, we elaborate on how the concepts are classified.

IV. Define the concepts and the concepts' hierarchy of STO

Defining the concepts and their hierarchies concern several approaches identified in the literature as mentioned in the guidelines of the proposed method. We selected the top-down approach assuming it would be more understandable by end users.

This definition engaged us with several steps as described briefly below:-

- 1) Categorise the main concepts according to the general classification as shown in Table 5.3:-

Table 5-3 Definition and general classification of STO

General Classification	Definition
Tester >	Terms include particular parties that conduct the test activity.
Task_Testing >	Terms include everyday jobs for performing the testing process.
Artefact >	Terms include related pieces to the test and testing process.
Environment >	Terms include the surrounding of the testing process and the trait terms from which the process can be described.

- 2) Identify the sub and sub-sub concepts of the high level concepts (the general classifications) as shown in Table 5.4:-

Table 5-4 Identifying the sub and sub-sub concepts of STO terms

Main Concept	Sub Concept	Sub-Sub Concept
Tester	Human	Individual
		Team
Environment	Software_Tool	
	Features	
	Hardware	
Artefact	Text	Code
		Document
		Data
		Measurement
	Images	
	Standard	Criteria
		Guide
		Report
		Plan
		Term
Task_Testing	Context	Purpose
		Scope
	Activities	Intrinsic
		Extrinsic
	Method	Technique
		Approach
		Practice

- 3) Classifying the remaining terms in the glossary into the identified concepts. For the full list of the classified terms, please refer to **Appendix A-1 STO Terms Classification**.

V. Define the properties of STO concepts

Usually concepts alone are not enough to give all necessary information. Hence, defining the properties to show the relations between different concepts in the ontology is a necessary step. As shown in Table 5.5, examples of relations between different concepts are identified.

The examples demonstrate a sample view, as STO was built within 59 different types of properties. For the full list of the properties, please refer to **Appendix A-2: STO Terms Properties**.

Table 5-5 Examples of Properties and their inverses

Concept	Object Property	Inverse Property
Individual	hasCheck	isCheckedBy
Team	hasControl	isContoledBy
Software_Tool	hasAutoProcess	isPerformedBy
Code	hasTest	isTestedBy
Data	hasResult	isResultsOf
Measurement	hasMeasurement	isMeasurementOf
Criteria	hasReview	isReviewedBy
Guide	hasStandard	isStandardFor
Plan	hasPlan	isPlannedBy
Term	hasModerate	isModerateBy
Purpose	hasPurpose	isPurposeFor
Scope	hasScope	isScopeOf
Extrinsic	hasPractice	isPracticeBy
Technique	hasTechnique	isTechniqueOf
Approach	has Approach	isApproachOf

VI. Define the data properties of STO concepts

This step required identifying the data type for each property. The benefit of using data type is the link which can be created between the classes and XML scheme. STO was built within 32 data type properties. Table 5.6 shows a sample of the data type. The domain field shows names of concepts that data represent, while the range shows the types of the data. For the full list of the data type, please refer to **Appendix A-3: STO Terms Data Properties**.

Table 5-6 Examples of Data Properties with their domain and range

Data Property	Domain	Range
hasTestID	Individual	string
hasNumberofLine	Code	integer
hasCreator	Artefact	string
hasStatus	Text	Boolean
isInfectedCode	Code	Boolean
hasSource	Test Case Suite	String
hasValue	Text	string
hasActualResulsts	Measurement	string
hasCriteriaDescription	Criteria	string
hasExpectedResutls	Text	string
hasPlanDescription	Plan	string
hasRatio	Feature	Null
hasSoftwareID	Null	string
hasGuidTitle	Guide	string
hasReportDescription	Report	string

VII. Create instances and individuals of classes

Once the concepts, properties and their data properties were defined, the last step in preparing STO is to create the instances and individuals of these concepts. Table 5.7 displays a sample of individuals. With regard to **STO** concepts, 106 individuals were built in. For the full list of the instances and individuals, please refer to **Appendix A-1 STO Terms Classification**.

Table 5-7 Examples of Concepts' Individuals

Class	Individual
Images	Call_Graph
	Cause-effect_Diagram
	Cause-effect_Graph
	Control_Flow_Graph
	Diagram
	Fishbone_Diagram
	Ishikaw_Diagram
	Mind-Map
	State_Diagram
Features	Accuracy
	Adaptability
	Availability
	Behavior
	Changeability
	Complexity
	Deviation
	Efficiency
	Executable
	Install-ability
	Maintainability

5.2 Implementation with PROTÉGÉ 4.0

We selected Protégé 4.0 as an open source standalone application, which is written in Java, and provides plug-and-play environment used for OWL editor to implement the **STO**. After the hard work to get the STO taxonomy ready as discussed in the previous section, we started the implementation by following the Protégé guide. Protégé with its plug in OWLViz provides a graphical view for the ontology. The graphic view makes it easy to understand the relations. Hence, we demonstrate the output of our process using the graphic view. The following steps demonstrate the accomplishments:-

I. Building the Classes Hierarchy

Classes are a concrete representation of concepts. We started building the classes to represent the STO taxonomy concepts. The following steps are described in detail as follows:-

- 1) Building parents classes to represent the general classification as shown in Figure 5.3

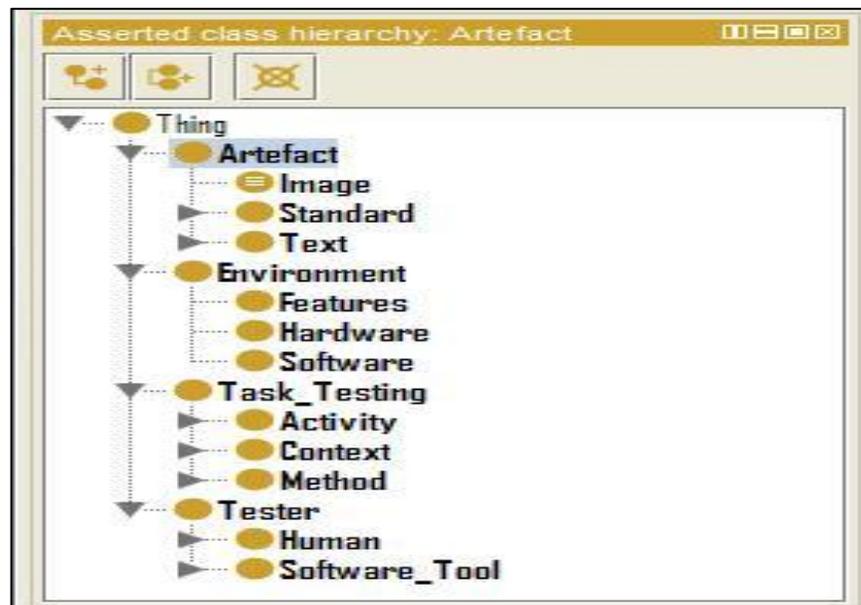


Figure 5-3 General Classes View for STO

2) Building children classes to represent the sub classes as shown in Figure 5.4

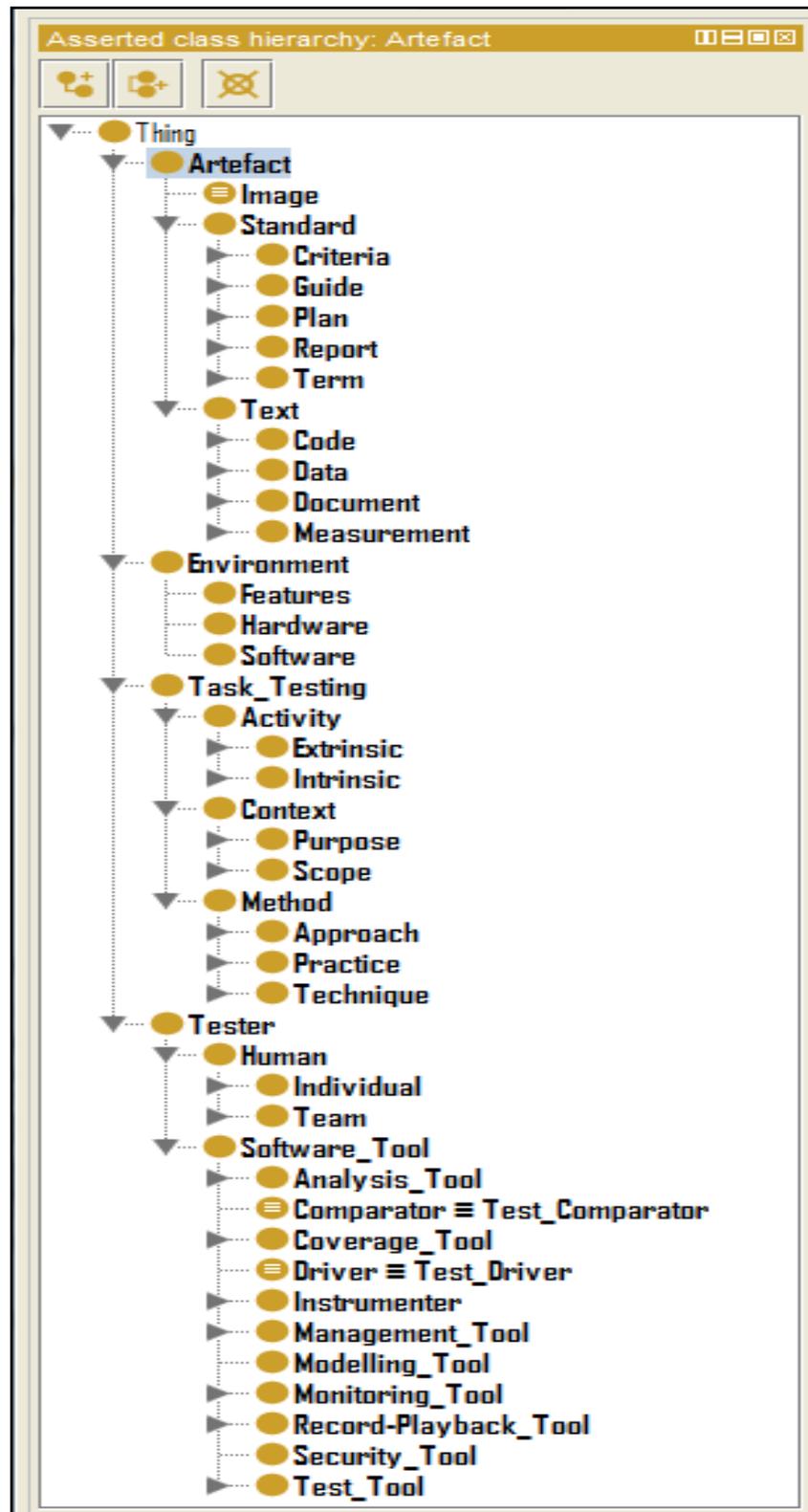


Figure 5-4 Sub Classes view for STO

3) Building grandchildren classes to represent the sub-sub classes as shown in Figure 5.5

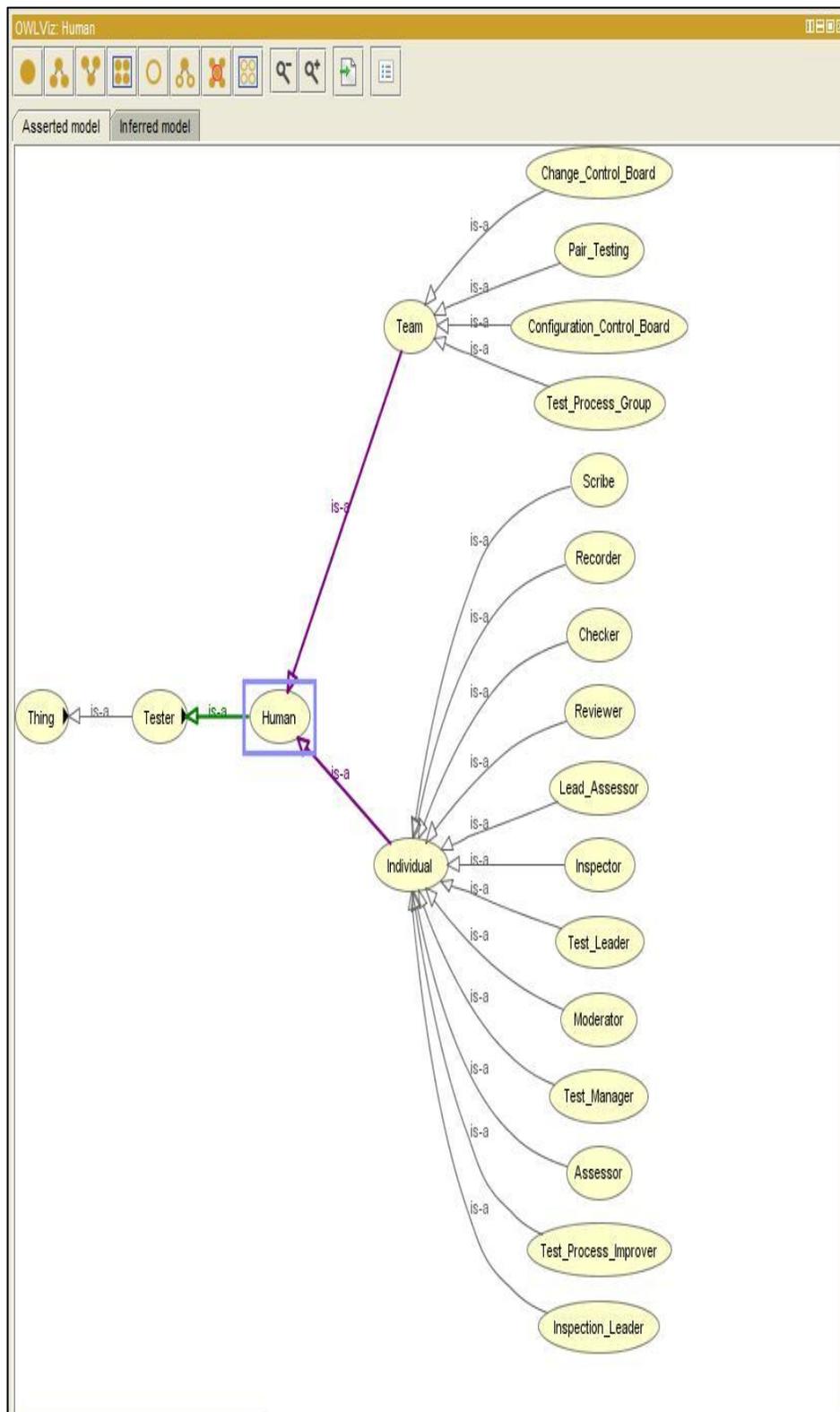


Figure 5-5 Sub-Sub Class view of STO

As shown, **STO** has four main layers. Each layer is described as follows:-

a) **Tester:**

This holds the meaning of what/who performs the task of testing. In this layer, Tester is either a person (i.e. human either individual or team) or software (i.e. tools for testing).

b) **Environment:**

This holds the meaning of related characteristics to Test. Environment has Features, Hardware and Software as subclasses.

- **Feature Class** comprises the behaviour terms such as (Pass, Fail and Testability, etc).
- **Hardware Class** comprises terms involving hardware such as (Sub, Storage, and Simulator etc).
- **Software Class** comprises the software terms such as (Buffer, System and Compiler, etc).

c) **Artefact:**

This holds the meaning of objects under the test activities. In the Artefact, we created Text, Image and Standard as subclasses.

- **Text Class** – all included terms describe the **Code, Document, Data or Measurement Data**.
- **Image Class** portrays instances of graphic terms in the domain.
- **Standard Class** includes all standards that have been inherited from standard organizations or frameworks. It is classified in **Guide, Criteria, Report, Plan or Term classes**.

d) **Task Testing:**

This defines terms of the main activities in the software testing domain that is in **Context, Activity or Method classes.**

- **Context Class** holds terms describing activities that occur in various software development stages, either for **Purpose or Scope.**
- **Activity Class** includes terms pointing to activities other than testing itself within (**Intrinsic**) or without (**Extrinsic**) the system.
- **Method Class** takes account of testing activities, whether it is a **Technique, Approach or Practice.**

Obviously with this simple explanation, the key factor that we depended on in building the general hierarchy classes of **STO** is to give effortless meaningful representation for a normal user with basic knowledge in software testing domain. Description Logic specifies hierarchy using restricted set of first-order formulas, and so does OWL reasoning rules. We defined a sub-set of OWL reasoning rules that support our hierarchy classes. For Instance, **Individual** class is illustrated in Table5.8:-

Table 5-8 STO hierarchy class rules

Rule	Description
subClassOf	$(?Individual \text{ rdfs:subClassOf } ?Human) \wedge (?Human \text{ rdfs:subClassOf } ?Tester) \Rightarrow (?Individual \text{ rdfs:subClassOf } ?Tester)$
disjointWith	$(?Individual \text{ owl:disjointWith } ?Team) \wedge (?Inspector \text{ rdf:type } ?Individual) \wedge (?Change_Control_Board \text{ rdf:type } ?Team) \Rightarrow (?Inspector \text{ owl:differentFrom } ?Change_Control_Board)$

II. Building the Object Properties

Object Properties are binary relations between the classes. After finishing building all classes, we created the possible relations (Object Property) between these classes. Figure 5.6 illustrates the object properties:

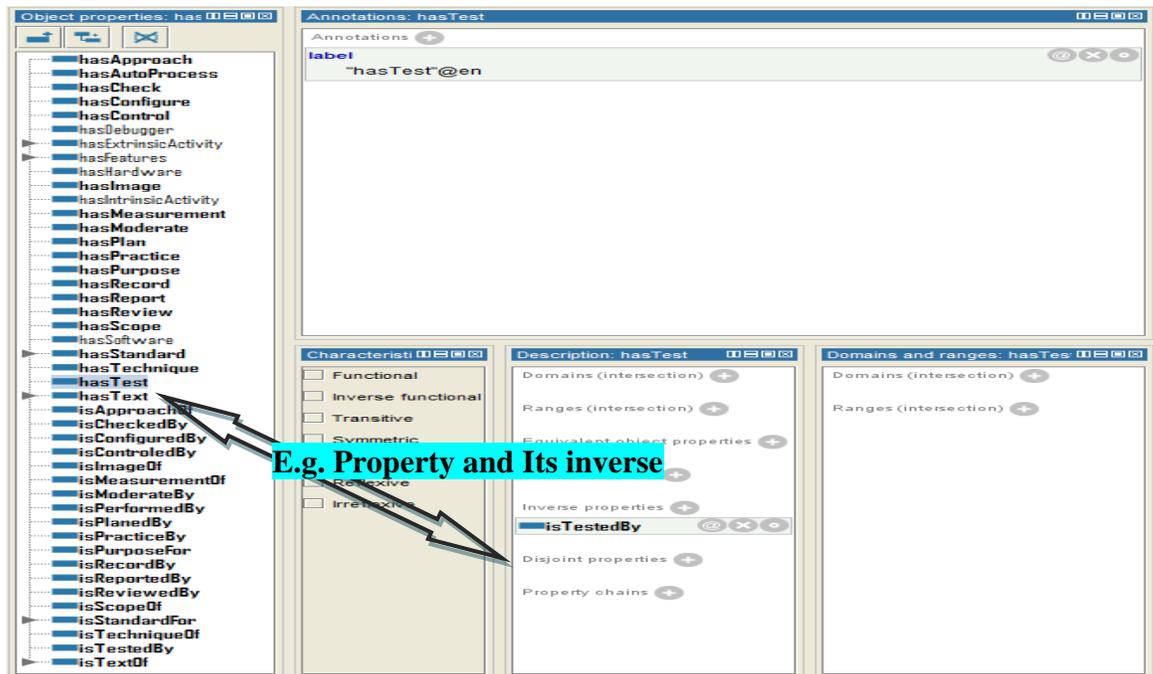


Figure 5-6 Object Properties View of STO

We defined a sub-set of OWL reasoning rules that support our object properties. For instance, **hasText & hasTest** property are illustrated in Table 5.9.

Table 5-9 STO property rules

Rule	Description
subPropertyOf	$(?hasDocument \text{ rdfs:subPropertyOf } ?hasData) \wedge (?hasData \text{ rdfs:subPropertyOf } ?hasText) \Rightarrow (?hasDocument \text{ rdfs:subPropertyOf } ?hasText)$
inverseOf	$(?hasTest \text{ owl:inverseOf } ?isTestedBy) \wedge (?Tester \text{ ?hasTest } ?Code) \Rightarrow (?Code \text{ ?isTestedBy } ?Tester)$

III. Building the Classes Data Properties

Data properties describe relationship between classes and data values. Some STO classes can be represented by data values. For instance, a test case needs to be represented by an ID or a Software Tool needs to contain a version to be traced. Hence, we created the data properties as shown in Figure 5.7.

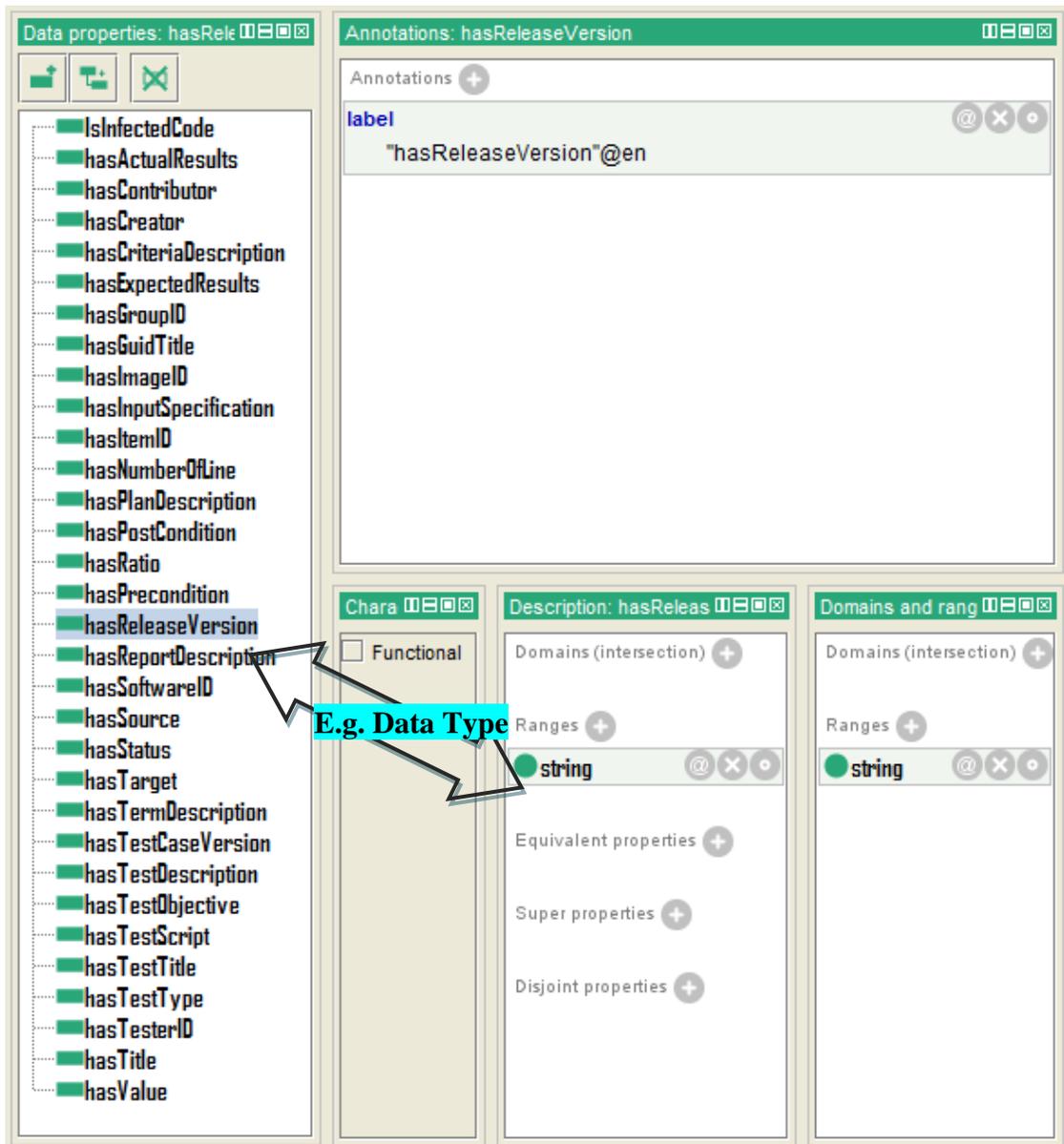


Figure 5-7 Data Properties View of STO

IV. Building the Classes' Individuals

Individuals represent objects in the domain. For instance, **Oracle** is an object term for **Software** class in the **Environment** concept in **STO** domain. Figure 5.8 demonstrates the examples of individuals, which had been built in STO.

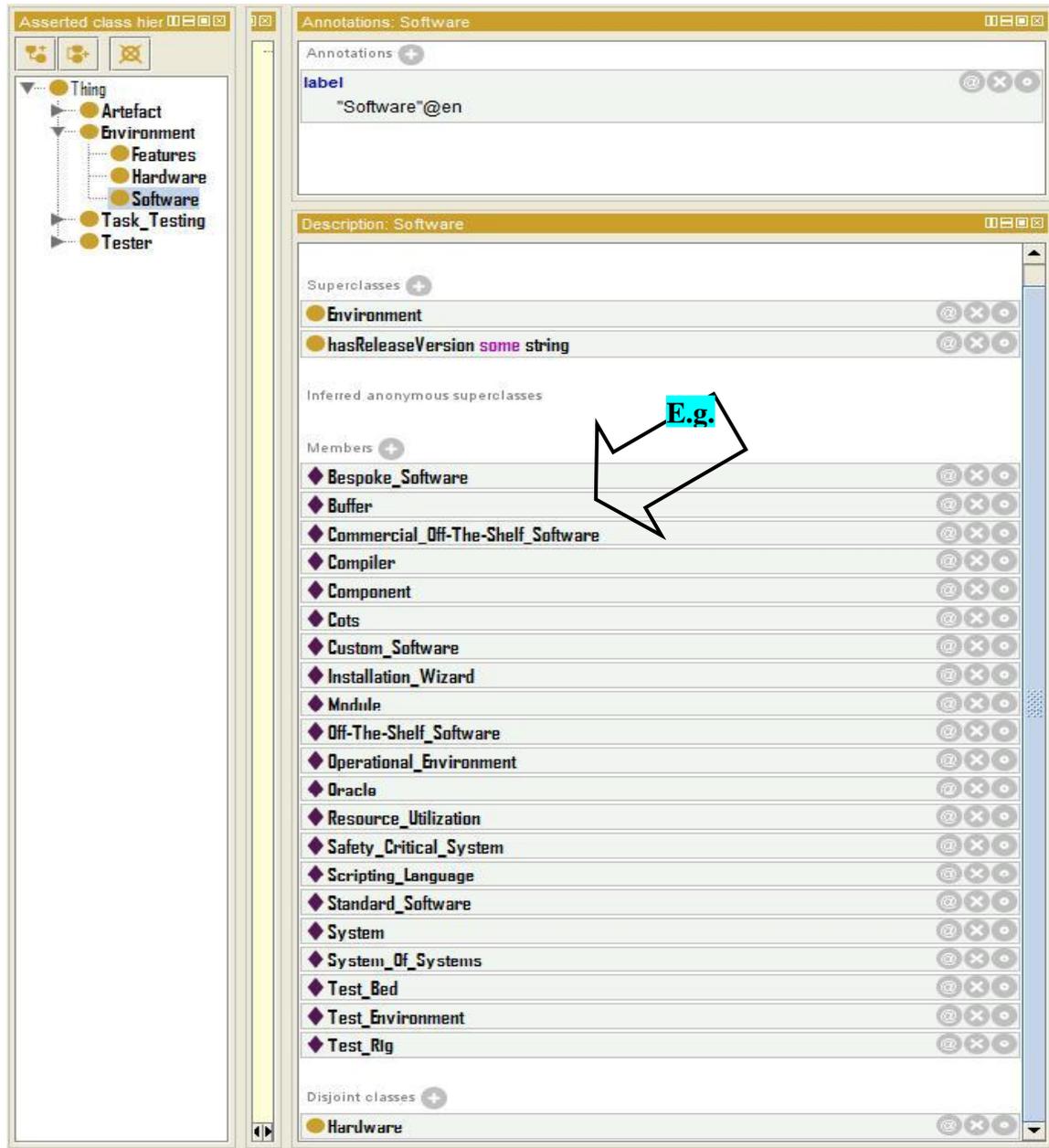


Figure 5-8 Individuals' view of STO

V. Building OWL Restrictions Rules

A restriction describes a class of individuals based on the relationships that members of the class participate in. **STO** restrictions are illustrated as follows:-

1) **Property Restrictions** which consist of:-

a) **someValuesFrom** –

Existential Restrictions are also known as *Some Restrictions*, or as *some values from restrictions*. For instance, Figure 5.9 demonstrates the **some** restriction for the Test Case class.

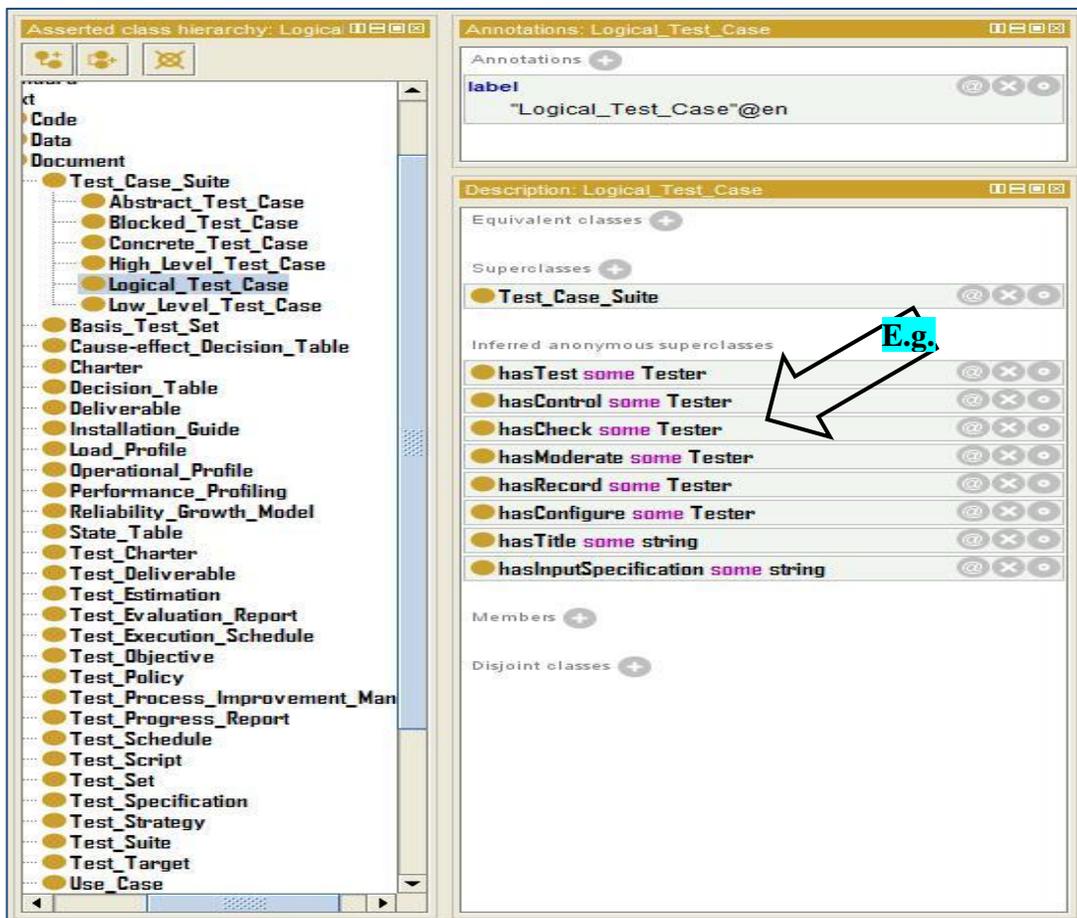


Figure 5-9 STO Some Values From restriction

It can be denoted in DL-Syntax as:

\exists hasTest Tester

b) **allValuesFrom** –

Universal Restrictions *are also known as all values from restrictions*. For instance,

Figure 5.10 demonstrates the **only** restriction for the Task Testing class.

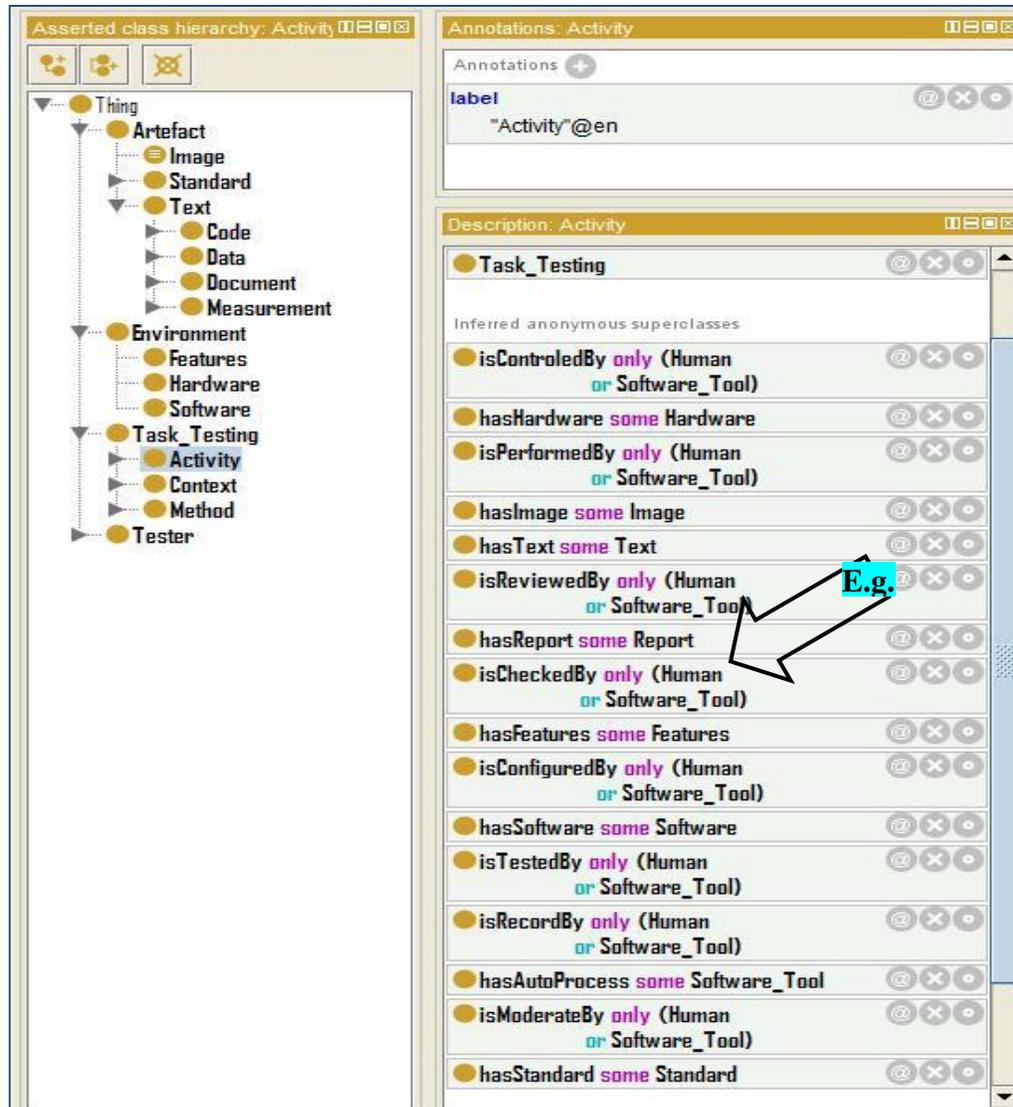


Figure 5-10 STO all Values From restriction

It can be denoted in DL-Syntax as:

\forall isCheckedBy Human or Software_Tool

2) **Data Restrictions** - A datatype property can also be used in a restriction to relate individuals to members of a given datatype. For instance, we demonstrate the **Code** class that has a **Boolean** data type to check if infected with bugs, has a **String** data type to carry the name of the code creator and **Integer** data type to store the number of codes as shown in Figure 5.11.

The screenshot displays a software interface with two main panels. The left panel, titled "Asserted class hierarchy: Code", shows a tree structure of classes. The right panel, titled "Annotations: Code", shows the annotations for the "Code" class. The "Annotations" panel includes a "label" annotation with the value "Code"@en. The "Description: Code" panel shows the class's properties and restrictions. A red arrow points to the "hasNumberOffline some integer" property, with the text "E.g." next to it.

Asserted class hierarchy: Code

- Thing
 - Artefact
 - Image
 - Standard
 - Text
 - Code
 - Data
 - Document
 - Measurement
 - Environment
 - Features
 - Hardware
 - Software
 - Task_Testing
 - Activity
 - Context
 - Method
 - Tester

Annotations: Code

Annotations +

label "Code"@en

Description: Code

Equivalent classes +

Superclasses +

- Text
- IsInfectedCode some boolean
- hasCreator some string
- hasNumberOffline some integer

Inferred anonymous superclasses

- hasTest some Tester
- hasControl some Tester
- hasCheck some Tester
- hasModerate some Tester
- hasRecord some Tester
- hasConfigure some Tester

Members +

Disjoint classes +

- Document, Measurement, Data

Figure 5-11 STO Data restriction

5.3 Summary

In this chapter, we presented the outcome of each step which has been followed to build the ontology for Software Testing domain. We emphasize again that STO presents concepts and terms of Software Testing domain based on a standard up-to-date glossary. These concepts and terms are linked in a formal structure. The formal structure of the STO consists of 626 Classes linked with 60 Object properties, identified by 32 Data properties and instanced by 106 Individuals. STO is a goal for developing Semantic Web for Test Case Management System. To build STO we followed the 101 guide for developing ontology, used the standard web ontology language recommended by W3C-OWL, and selected Protégé version 4.0 as a tool to implement it.

6.0 Implementation of Semantic Test Case Management System

Semantic Test Case Management System (STCMS) is the first Test Case Management System that implements Software Testing Ontology. The uniqueness of the system are the illustration of Well-Structured Test Case as individuals with comprised management information, which is represented semantically, and the integration of the Software Testing Ontology to facilitate the management testing process.

STCMS allows the users to create, store, retrieve and update test cases using semantic technology. It also implements automate information retrieval for terminologies and taxonomies of software testing domain based on explicit conceptual hyperlinked relations. It enhances the traditional search results (which is based on word occurrence). The requirements of STCMS were gathered from different perspectives. We initially used the literature and relevant work in chapter 2. Secondly, they were observed by studying other testing management systems' requirements.

Following the aforementioned ways of gathering requirements, we are able to come out with functional and non functional requirements for the STCMS, which are specified and documented in Software Requirement Specification IEEE standard. The IEEE standard was tailored to fit our required template. Then the system use cases are created according our template.

The system development lifecycle adopted the software engineering disciplines using the Rational Unified Process (RUP) as it provides structured and well-controlled methods. The development relies on the use of Unified Modelling Language (UML) on

modelling the modules of the system. The system is a web application solution, which uses JSP as a front end and My-SQL as a back end. Furthermore, the system uses Java language for the module implementation, Jena API as it is a full-feature Java for RDF and SPARQL as a special query language and suitable for RDF. For the requirements' specification and details design of the system, see **Appendix B-1 SRS & B-2 SDD**.

6.1 Requirement

The main objective here is to achieve the ability of representing and searching semantically the individual test cases. This requirement includes the following:

1. Automation process for sharing and reusing test cases for computer to manipulate.
2. Effective and efficient facilitation of the testing process by providing well structured test cases linked to other testware and software artefact.
3. Minimising cost and maximizing efficiency by providing a semantic search.
4. Software Testing information retrieval to provide a component that can be utilized by third party (regardless of machine or human) to not only explore the term, but be able to pull all relevant data for that term

6.2 Test Case Collection

Test case documents are considered as archival data, which is a third degree level of data collection technique. In this technique, as the data is not developed with the intention to provide data for the research (STCMS in our case), the quality may be affected (Runeson & Höst, 2009).

To overcome this issue in the test case collection for our research case study, we developed a Test Case Template. Moreover, there were several different sources for collecting the test cases. The followings describe the process of STCMS test case collection.

6.2.1 Test Case Template

We formulated a template for the required parties to fill the mandatory data for our research to constitute a standard format. The Template is shown in Table 6.1. Industry testers are overloaded with too much work. Hence, we kept the mandatory data required to fill in the template and provided description for each field as follows:-

1. **System:** The acronym of the system's name for the test case to test (e.g. Semantic Test Case Management System – STCMS)
2. **Test Case No.:** The identification number for the test case plus the name of the test case (e.g. 1. Login)
3. **Test Case Version:** The version of the test case was assigned with Version 1.0 since it was created for the first time in our case study.
4. **Test Title:** A unique title which starts with the acronym of the system's name and the name of the test case (e.g. STCMS_Login)
5. **Test Objective:** The objective to conduct the test case (e.g. To check whether the entered User name and Password are valid or Invalid)
6. **Pre-requisite:** The precondition of the test case (e.g. The web site is uploaded and the user is registered)
7. **Tester:** The name of the creator of the test case (e.g. Mansoor)

8. **Step:** The index of the input required (e.g. 1.2.3.or I.II.III)
9. **Description:** The input procedures for the test case to be followed (e.g. 1. Actor enters username/ password and click sign in)
10. **Expected Result:** The expected reaction from the tested system after each input (e.g. System generates error message/ **Invalid ID or password Please try again**)

Table 6-1 Test Case Template for Collecting Data

System:		
Test Case No.	1.	Test Case Version Version 1.0
Test Title		
Test Objective		
Pre-requisite		
Tester		
Step	Description	Expected Result
1.		
2.		
3.		

6.2.2 Test Case Sources

We identified two sources – (1) Academic Prototype Systems and (2) Industry Company System to collect the test cases from. For the test cases please refer to **Appendix C**. The sources are discussed as follows:-

I. Academic Prototype Systems:

For this source, we gathered test cases from two prototype systems as follows:-

- ✓ **STCMS:** We created test cases to validate the functionality of our system and used the test cases as sample data to run the system, refer to **Appendix B-3 STD**.
- ✓ **FSKTM PERSONALIZED WEBSITE (FPW):** is Faculty of Computer Science and Information Technology users' personalised website. For the purpose of giving support, our lab colleagues who had developed this prototype provided us with their test cases, refer to **Appendix C-1**.

II. Industry Companies System:

For this source, we contacted several companies based in Malaysia. The selection of the companies was based on their willingness to share and publish the test cases in the research thesis. Those who gave a positive response are discussed below:-

- ✓ **i-Cognitive Software Solution:** Provided us with iLogger System test cases. iLogger does health checks on the machine performance without requiring human to monitor, refer to **Appendix C-2**
- ✓ **Sapura Secured Technologies:** Provided us with two Systems test cases - FoodReg, a web-based application, refer to **Appendix C-3**

6.3 STCMS Discussion

STCMS is a Test Case knowledge management system that has been deployed in a Semantic Web-based environment. It captures and represents not only test cases, but also other related testware and software artefact with the association of metadata into test case structure, as well as the provision of ontology to harness the real power of the semantic representation.

In the previous sub-headings, we described the design process of what we call Well-Structured Test Case, and we presented how we collected the test cases data. In this sub-heading and for the purpose of showing how we developed the system and used the test case data, we demonstrate the component architecture and features of the system, then brief on inserting the test cases to run the system.

The process undertaken to develop STCMS culminated in the following procedures:-

I. Component Architecture

In this process, we developed the modelling diagram, which describes the main components of STCMS using the Unified Modeling Language (UML) as shown in Figure 6.1. The architecture shows the STCMS's platform-independent. The main features of the component are described in the following sub-section.

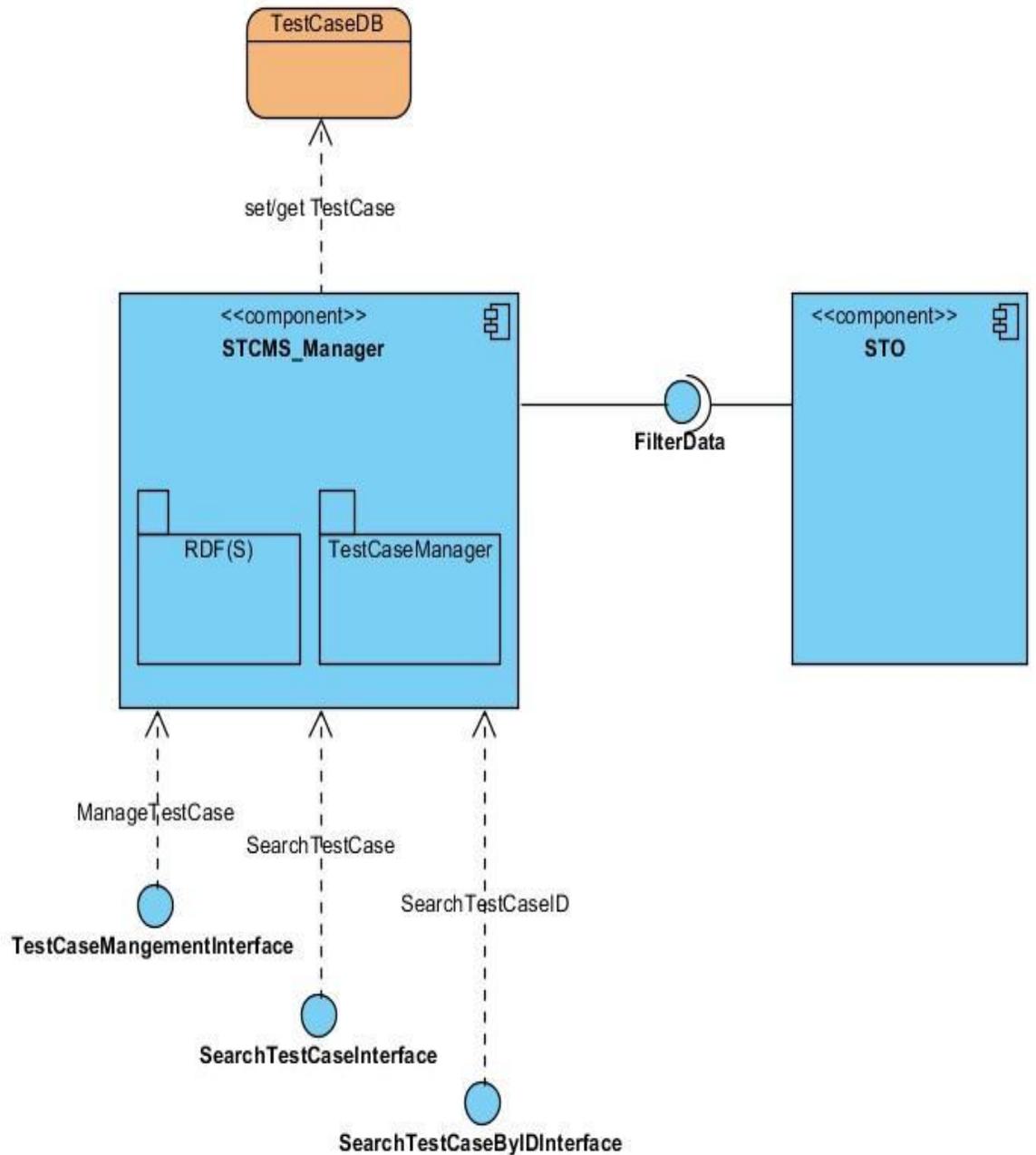


Figure 6-1 STCMS' Component Architecture

II. Features:

The main features were implemented are reflecting the research objectives. They are illustrated as follows:-

a) **Test Case Management:** here, we facilitate the following functions:-

- ✓ **Create** the test cases based on RDFS using the Jena API. The created test case is saved in MySQL database as shown in Figure 6.2.

The screenshot shows a web browser window with the URL `http://localhost:8080/SWbSTS_Main_IV`. The browser's address bar shows the page title as 'JSP Page'. The page has a red navigation bar with links: Home, Create TestCase, Search TestCase By ID, Search TestCase, View TestCases, and Software Test. The main content area is titled 'Create Test Case Form' and contains the following fields:

- ID:
- Description:
- Creator:
- Purpose:
- Source:
- Subject:
- Relation:
- TestType:
- [Add Input](#)
- Input 1: Expected result1:
- Input 2: Expected result2:

At the bottom of the form are two buttons: 'Create Test Case' and 'Reset'. A callout box with a white background and black border points to the 'Add Input' link and the input/result fields, containing the text 'Additional inputs for test case'. At the bottom of the page, there is a copyright notice: 'Copyrights@ManoorHak-2010'.

Figure 6-2 Create Test Case Form

- ✓ **View** the test cases based on RDFS using the Jena API. The stored test case will be retrieved from the database, and displayed and represented for the user as shown in Figure 6.3.

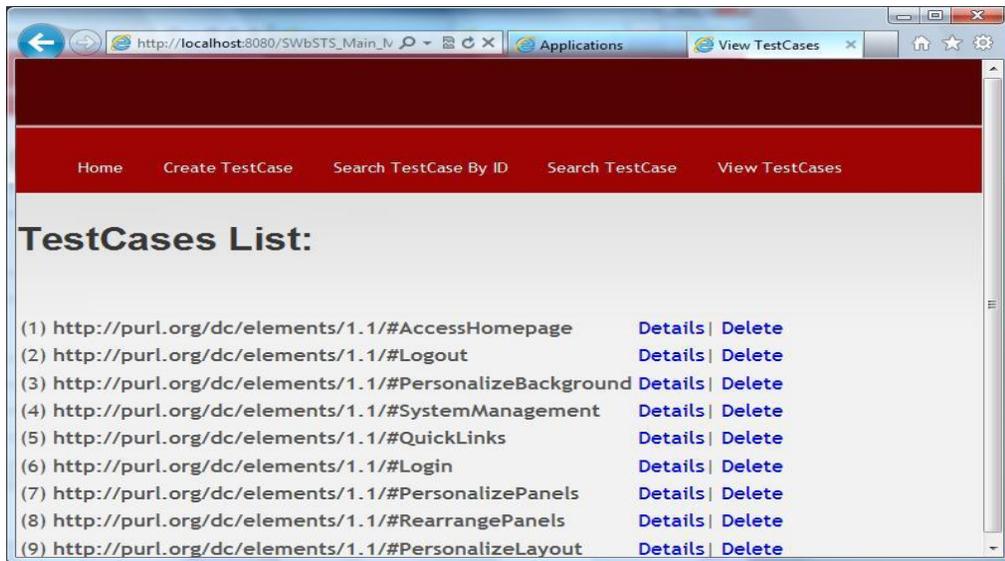


Figure 6-3 View Test Case

- ✓ **Edit &Delete** the test case features are provided in the View test case list as shown in Figure 6.4.

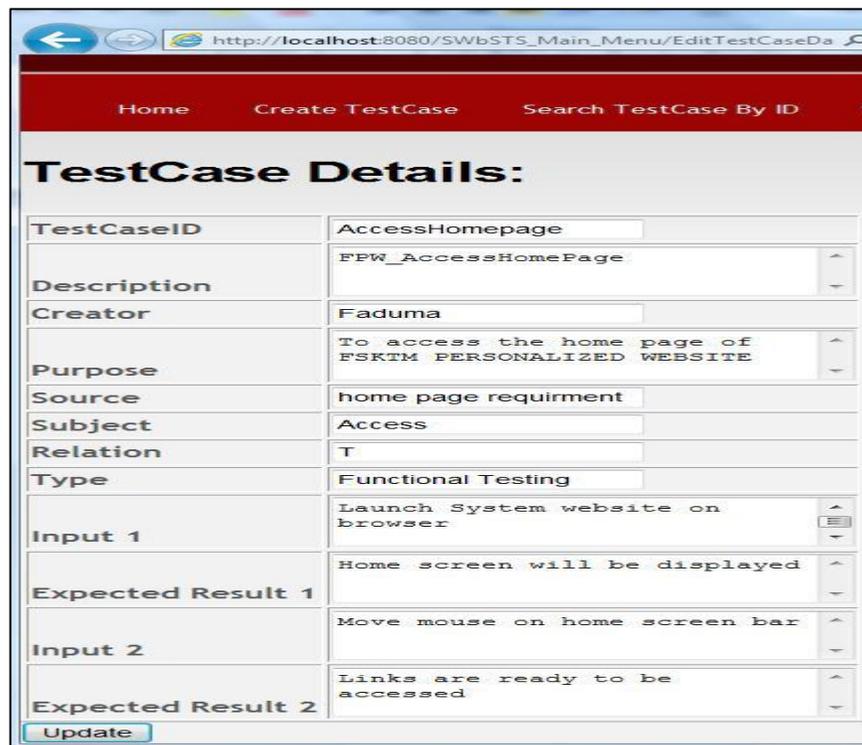


Figure 6-4 Edit Test Case

- b) **Test Cases Semantic Search:** the facility here is to search for test cases. The semantic search feature assists the user (regardless of human or machine) with dynamic terms that match the search terms if available in the STO. Furthermore, there is a Navigation Bar, which displays all the available concepts belonging to the searched term as shown in Figure 6.5.

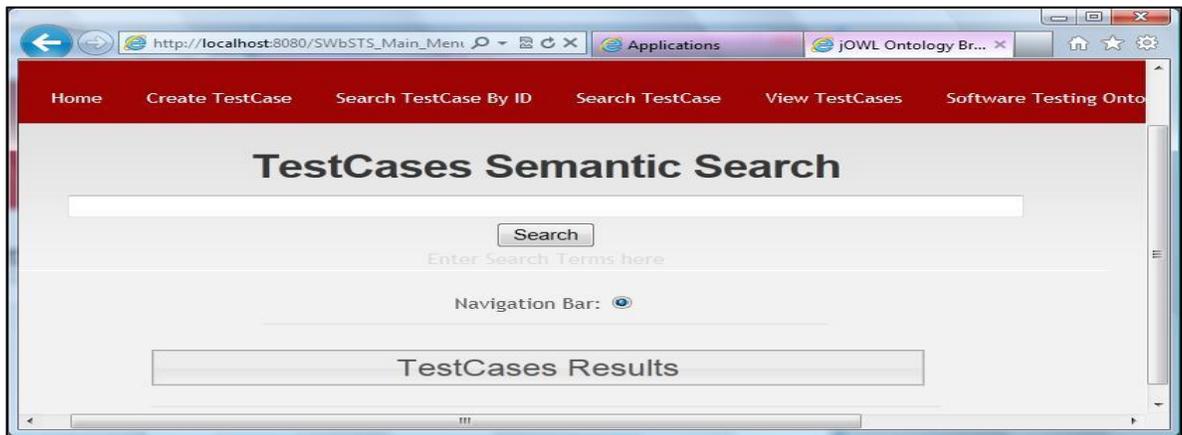


Figure 6-5 Semantic Search Form for Test Cases

- c) **Search Test Case by ID:** This keyword search feature is just to ease the process of finding the test cases if the ID is known to the user as shown in Figure 6.6.



Figure 6-6 Search Test Case by ID

- d) **Class View:** provides all **STO** conceptual terms in a hyperlink and search text field, where a user can find the hierarchy, the related descriptions and all possible relations of the search term as shown in Figure 6.7.

The screenshot displays the STO Class View interface. At the top, there is a navigation bar with links: Home, Create TestCase, Search TestCase By ID, Search TestCase, View TestCases, Software Testing Ontology, and Resources. Below this, there are tabs for Classes, Properties, Individuals, and SPARQL-DL. The main content area lists various classes as hyperlinks, including Abstract Test Case, Acceptance Criteria, Activity, Actual Outcome, Actual Result, Analysis Tool, Analyzer, Approach, Artefact, Audit Trail, Automated Testware, Baseline, Basic Block, Basis Test Set, Benchmark Test, Best Practice, Branch, Branch Condition, Branch Condition Coverage, Branch Coverage, Bug Report, Bug Trace, Capability Maturity Model, Capability Maturity Model Integration, Capture-Playback Tool, Capture-Replay Tool, Cause-effect Decision Table, Certification, Change Control Board, Checker, Chows Coverage Metrics, Classification Tree, Code, Code Analyzer, Comparator, Completion Criteria, Component Specification, Compound Condition, Concrete Test Case, Condition Combination Coverage, Condition Coverage, Condition Determination Coverage, and Configuration. A callout box with an arrow points to a link in this list, containing the text "By Clicking the link". Below the list is a "[more]" link. Underneath, there are radio buttons for "Treeview" (selected) and "Navigation Bar". The interface is split into two panels: "Treeview" on the left and "Description of Abstract_Test_Case" on the right. The Treeview panel shows a hierarchy: Artefact > Text > Document > Test_Case_Doc > Abstract_Test_Case. The Description panel shows "Terms: Abstract Test Case, Abstract_Test_Case" and "Relations" with properties like hasConfigure, hasControl, hasModerate, hasRecord, hasCheck, and hasTest, each followed by a "Tester" link. A callout box with an arrow points to the "Abstract_Test_Case" entry in the treeview, containing the text "Or Searching the terms". At the bottom left, it says "Created by Mansoor Hak".

Figure 6-7 STO Class View

- e) **Property View:** provides all **STO** object properties in a hyperlink, where a user can find the related description of the selected object as shown in Figure 6.8.

Home Create TestCase Search TestCase By ID Search TestCase View TestCases Software Testing Ontology Resources

Classes Properties Individuals SPARQL-DL

ObjectProperties

[hadData](#), [hasApproach](#), [hasAutoProcess](#), [hasCertification](#), [hasCheck](#), [hasCode](#), [hasConfigure](#), [hasControl](#), [hasCriteria](#), [hasDocument](#), [hasExtrinsicActivity](#), [hasFeatures](#), [hasHardware](#), [hasImage](#), [hasIntrinsicActivity](#), [hasMeasurement](#), [hasModerate](#), [hasPractice](#), [hasPurpose](#), [hasRecord](#), [hasReference](#), [hasResult](#), [hasReview](#), [hasScope](#), [hasSignificance](#), [hasSoftware](#), [hasStandard](#), [hasTechnique](#), [hasTerminology](#), [hasTest](#), [hasText](#), [isApproachOf](#), [isCheckedBy](#), [isCodeOf](#), [isConfiguredBy](#), [isControlledBy](#), [isCriteriaOf](#), [isDataOf](#), [isDocumentOf](#), [isImageOf](#), [isModerateBy](#), [isPerformedBy](#), [isPracticeBy](#), [isPurposeFor](#), [isRecordBy](#), [isReferenceOf](#), [isResultOf](#), [isReviewedBy](#), [isScopeOf](#)

[more]

Shows the description of the property for

Description of isDocumentOf

Terms: is Document Of, isDocumentOf

Created by Mansoor Hak

Figure 6-8 STO Properties View

- f) **Individual View:** provides all **STO** individuals in a hyperlink, where a user can find the related concept of the selected individual as shown in Figure 6.9.

Home Create TestCase Search TestCase By ID Search TestCase View TestCases Software Testing Ontology Resources

Classes Properties Individuals SPARQL-DL

[Accuracy](#), [Adaptability](#), [Analyzability](#), [Anomaly](#), [Availability](#), [Changeability](#), [Complexity](#), [Efficiency](#), [Error Tolerance](#), [Executable](#), [Exercised](#), [Fail](#), [Fault Tolerance](#), [Functionality](#), [Incident](#), [Interoperability](#), [Memory Leak](#), [Mistake](#), [Operability](#), [Portability](#), [Probe Effect](#), [Replaceability](#), [Robustness](#), [Safety](#), [Scalability](#), [Software Test Incident](#), [Stability](#), [State Transition](#), [Test Fail](#), [Test Incident](#), [Test Pass](#), [Test Session](#), [Testability](#), [Traceability](#), [Understandability](#), [Usability](#), [Attractiveness](#), [Behavior](#), [Buffer Overflow](#), [Bug](#), [Co-Existence](#), [Compliance](#), [Defect](#), [Deviation](#), [Error](#), [Failure](#), [Installability](#), [Learnability](#), [Maintainability](#)

Shows the main class

... [more]

Description of Test_Session

owl:Class Features

Created by Mansoor Hak

Figure 6-9 STO Individual View

- g) **SPARQL-DL**: provides users with a query text area for entering SPARQL-DL syntax to query the **STO** as shown in Figure 6.10.

The screenshot shows a web application interface for querying the Software Testing Ontology (STO). At the top, there is a navigation bar with links: Home, Create TestCase, Search TestCase By ID, Search TestCase, View TestCases, Software Testing Ontology, and Resources. Below this is a tabbed interface with 'Classes', 'Properties', 'Individuals', and 'SPARQL-DL' selected. The main area is titled 'Enter a query:' and contains a text input field with the query: `SubClassOf(?sc, Test_Case_Dod)`. Below the input field, there are two columns: 'Results' and 'Syntax'. The 'Results' column shows a list of 7 results: `?sc : Abstract_Test_Case`, `?sc : Blocked_Test_Case`, `?sc : Concrete_Test_Case`, `?sc : High_Level_Test_Case`, `?sc : Logical_Test_Case`, `?sc : Low_Level_Test_Case`, and `?sc : Test_Case_Suite`. The 'Syntax' column lists various SPARQL-DL constructs: `Type(?a, ?b)`, `PropertyValue(?a, ?b, ?c)`, `Class(?a)`, `ObjectProperty(?a)`, and `DatatypeProperty(?a)`. A note below the syntax list says 'Examples: see jOWL SPARQL-DL Test Suite page'. Two callout boxes are present: one pointing to the results list with the text 'Shows the results of the class of Test Case', and another pointing to the syntax list with the text 'Examples of SPARQL-DL query syntax to help users'. At the bottom left, it says 'Created by Mansoor Hak'.

Figure 6-10 STO Query View

III. Insert Test Cases:

In this process, we inserted the aforementioned test cases in the test case Collection sub-heading. With the amount of the test cases collected, we were able to run the system and the results are discussed in the result evaluation chapter 7.

6.4 Limitation

Having completed the discussion above, we still believe there are more to be done and we consider them as out of our case study scope. Nevertheless, they are limitations for STCMS and can be implemented in future work. From our point of view, they include the following:-

I. Integration with other Database:

A limitation in STCMS is not having the relevant feature to integrate with other existing systems to restructure their test cases. Currently, users need to create test cases manually or key in to the system. Adding the integration feature to add the capability to read from other systems' databases or auto reading from text files will facilitate the reuse of existing test cases without the burden of creating or keying in to the STCMS.

II. Complex-Structure Test Case:

This can be counted as another limitation. We refer to the work that had been done by Christophe Strobbe et al. (2006) and C. Strobbe & Velasco (2005), where they linked test cases to Test Suits, which results in having more required elements for the test case structure. Our research focuses on dealing with individual test cases to be represented. Hence, future work for STCMS could be upgraded to dealing with group test cases that build Test Suits.

III. Ontology Update Interface:

It is a feature that might assist the system in updating the STO to give more to-date terms in semantic search. This feature can serve as a manual process at the beginning, and can be automated in the future

6.5 Summary

In this chapter, we presented the implementation of the Semantic Test Case Management System STCMS. STCMS is an ontology-based application that is implemented using semantic technology concepts and featured with the management process (create, view, edit, delete and search). At present, STCMS represents 51 test cases of different systems, which can be scaled up to be within the limitation of storage capability. Moreover Well-Structured Test Case attributes and metadata are designed and presented in RDFS. This representation makes it possible for the retrieval of test case for reusability purposes and machine manipulation. Finally the automatic information extraction on STO is a gate to search up-to-date Software Testing Terms in an efficient way to help users locate the matching terms to minimize their search efforts.

7.0 Evaluation

We have argued that the work presented in this thesis is novel in the following aspects: Software Test Ontology, automated software testing information extraction, representing well-structured individual test cases and test case Semantic Search. The STO represents the conceptual connection between the software testing terms in STCMS. Therefore, STCMS is capable of extracting the testing information automatically from the STO. Moreover, representing individual test cases in RDFS makes it easier for searching the test cases semantically for managing and reusing them. By evaluating the novelty presented in this work, we will prove the significance and benefits of the STCMS to the body of knowledge.

7.1 Evaluation Criteria

To determine whether objectives 3 and 4 described in section 1.4 are achieved, we designed the following evaluation criteria:

1. Correctness of Software Test Ontology using built-in reasoners, discussed in section 7.2.1
2. Proximity of the automated software testing information extraction using semantic similarity, discussed in 7.2.2
3. System Usability Scale (SUS) developed by Brooke (1996) to allow the practitioner to quickly and easily assess the usability of a given product or service.7.2.3
4. Performance of the semantic search using precision and recall, discussed in section 7.2.4

7.2 Evaluation Process

7.2.1 Evaluation of Software Test Ontology

STO is evaluated by using reasoning service offered by reasoners plugged in Protégé. The main benefits of the services are computing the classes' hierarchy and logical consistency checking. The STO verification process started at the early stages of the development to ensure the correctness and avoid propagation errors. We used two reasoners to verify STO as shown in Figure 7.1.

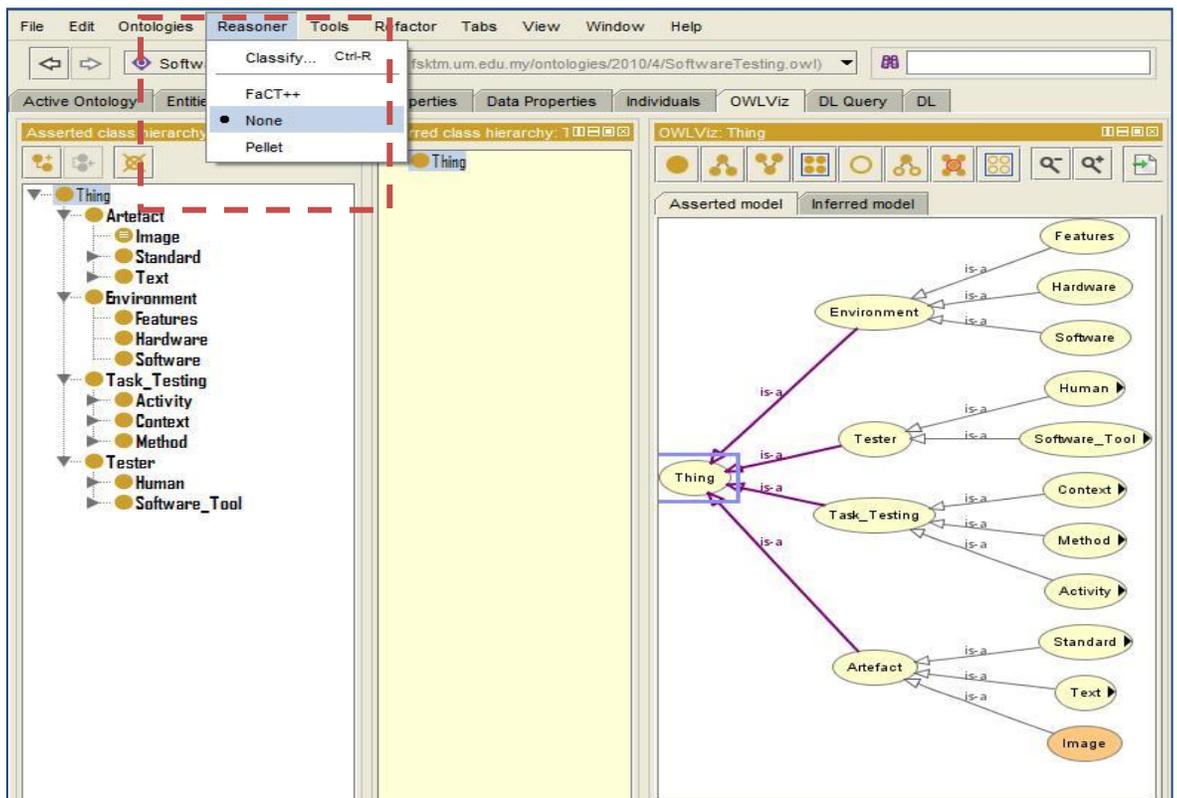


Figure 7-1 Reasoners Used to evaluate the STO

The task of computing the inferred class hierarchy is also known as classifying the ontology is described as follows:-

- I. FaCT++: the first reasoner was used as it is shipped with Protégé. The inferred hierarchy is the automatically computed class hierarchy by the reasoner. Figure 7.2 presents the inferred hierarchy graph showing the “no exists” of the inconsistent class. In case of inconsistencies, Protégé would highlight them in red. Meanwhile, the class “Nothing” is to identify the inconsistent classes if any exist.

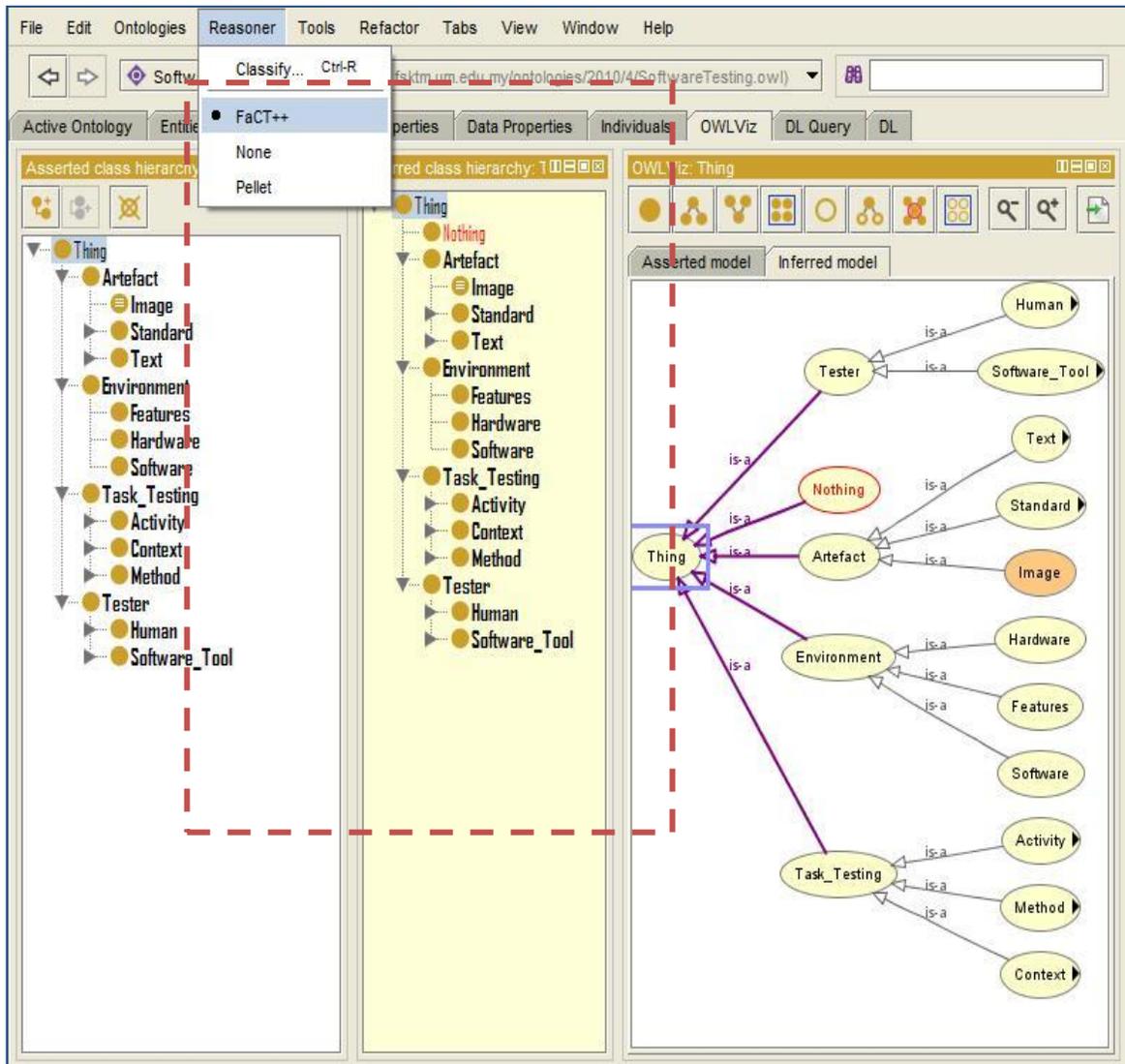


Figure 7-2 FaCT++ “Nothing” class shows the “no exists” of Inconsistent Class

II. Pellet: the complete OWL-DL reasoner (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007). Protégé allows Pellet plug-in to be installed and compute the OWL. Hence, we computed STO via Pellet for a second evaluation. Figure 7.3 presents the inferred hierarchy graph showing the “no exists” of inconsistent class.

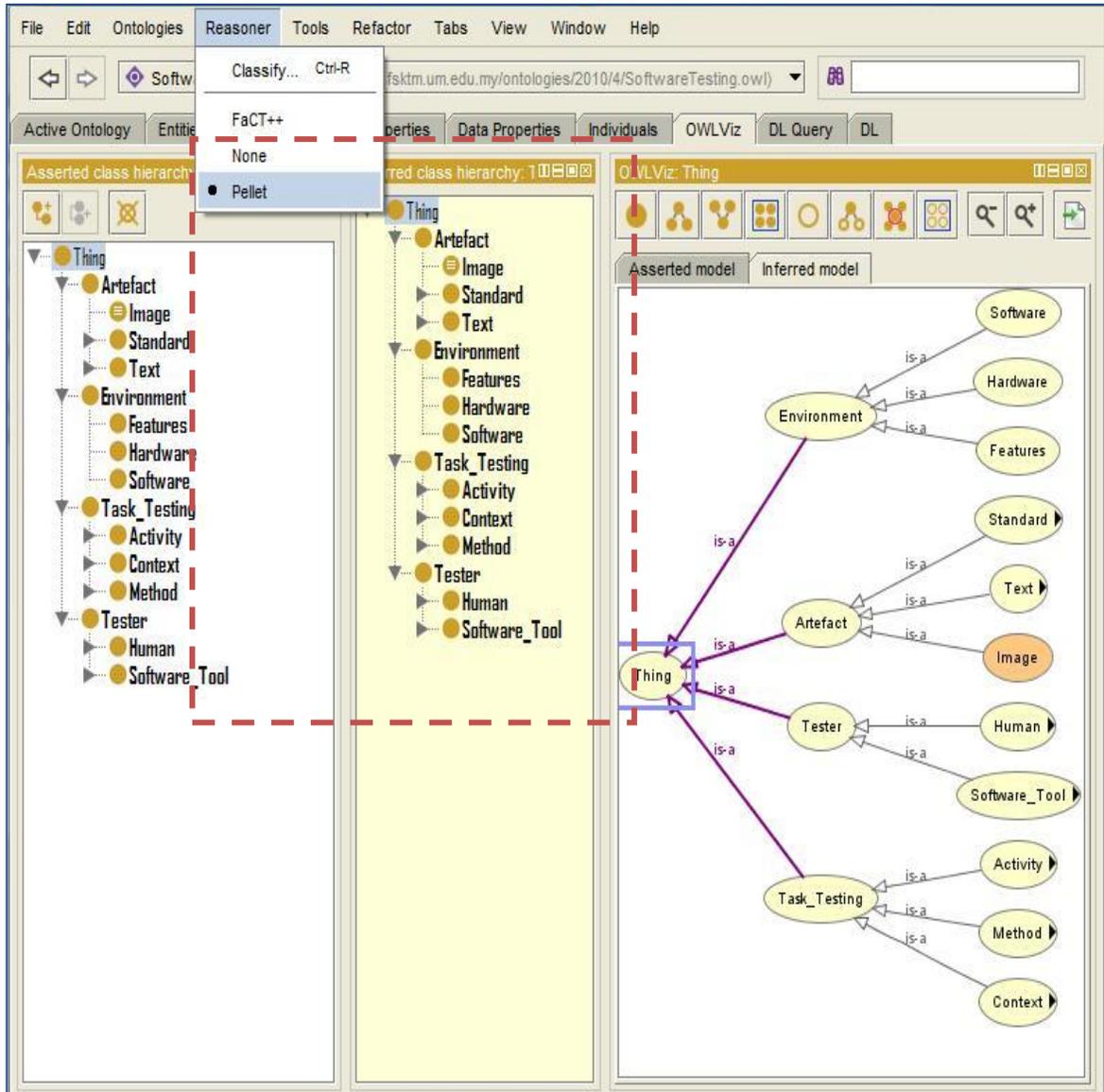


Figure 7-3 Pellet reasoner shows the “no exists” of Inconsistent Class

7.2.2 Semantic Similarity

Our STCMS provides context aware query capability. Firstly, we allow users to insert request in their natural language (English as default). Then, we automatically process and match on-the-fly the request with our semantic indexing. The matching performs the actual comparison between the request and the semantic index. The related information is then retrieved and displayed in the user browser.

For instance, the concept **Method** has been randomly selected to be examined. The GUI interface shown in figure 7.4 transforms the free-text query into the semantic representation. On-the-fly matching retrieves and displays the related information to the requested query.

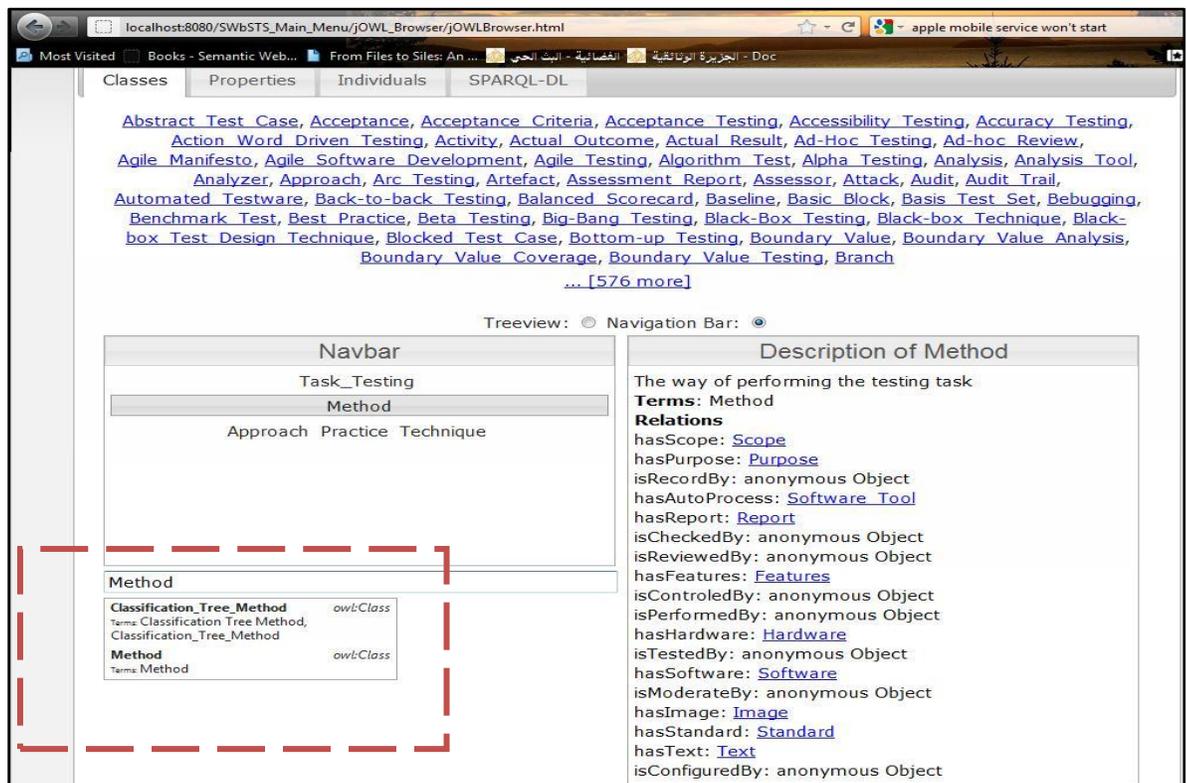


Figure 7-4: GUI transforms the free-text query into the semantic representation

In order to evaluate the proximity of our matching results, we use the semantic similarities adopted from Lin's (1998). It refers to the similarity between the corresponding generic concepts of the query term and results to show its precision. It is measured using the following formula:

$$Sim(x_1, x_2) = \frac{2 \times \log P(C_0)}{\log P(C_1) + \log P(C_2)}$$

where C is the generic concept in the ontology, x is \in from C and $P(C)$ is the randomly selected probability. C_1 & C_2 are independent concepts, while C_0 is the most specific concepts subsume them. In our experiments, we identified the following:-

1. matched concepts are similar when $Sim \rightarrow 1$
2. matched concepts are less similar when $Sim \rightarrow 0$

Following the instanced concept Method used earlier, Figure 7.5 illustrates a fragment of the STO where the concept Method connected with other concepts.

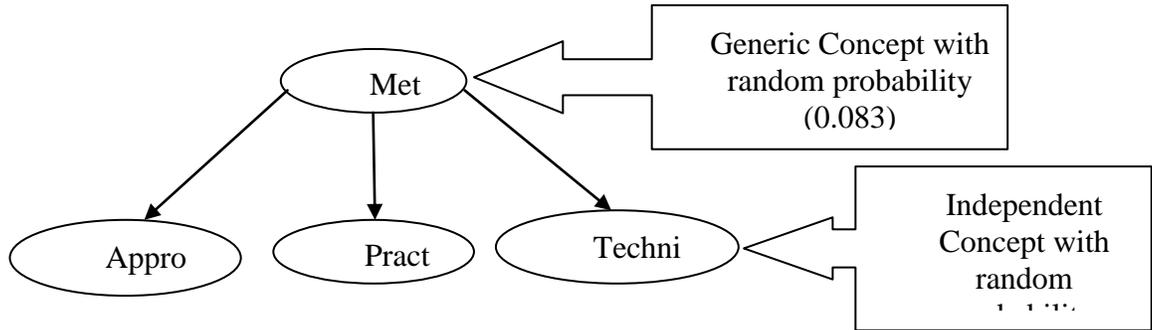


Figure 7-5 A Fragment of STO terms

For example Approach and Practice the semantic between these retrieved concepts are

$$Sim(Approach, Practice) = \frac{2 \times \log P(Method)}{\log P(Approach) + \log P(Practice)}$$

which is equal to 0.69.

Table 7.1 shows that similar concepts to the requested concept are automatically extracted. These concepts are semantically represented for the user as discussed in section 6.4. We observed that the matched concepts are semantically similar to the requested concept. The results show that our STCMS is capable of semantic information retrieval.

Table 7-1 Results of semantic similarity

x_1	Approach	Practice	Technique
x_2			
Method	0.69	0.69	0.69

7.2.3 Usability

I. User-based Usability Evaluation

STCMS was built in order to allow Testers to manage individual well-structured test cases. In order to know if the system is used easily and effectively, we evaluated the system usability as it correlates directly by the aforementioned reasons. In general the aim of measuring the usability of STCMS is to evaluate the systems' core features specifically the semantic search from the user's point of view.

The methodology we used to perform the experiment was to observe users in a session of the system. Users were given a period of time with STCMS and then asked to fill a questionnaire to express their views on the different features of the system. The questionnaire used for evaluating was driven from the System Usability Scale (SUS) (Brooke, 1996) as SUS is one of the most popular questionnaires containing a standardized collection of questions.

Measurements of usability have several different aspects:

- Effectiveness: Can users successfully achieve their objectives?
- Efficiency: How much effort and resource is expended in achieving those objectives?
- Satisfaction: Was easy to use the system?

The result of the questionnaire is a value between 1 and 100, where 1 signifies that a user found a system absolutely useless and 100 that a user found a system optimally useful.

We chose a total of 30 participants to perform the experiment. All participants were professional software engineers with variety years of experience who are familiar with system development process. Therefore, they were able to give us good feedback regarding the core features. We uploaded the system online during the testing period and then each participant was asked to navigate and go through each feature of the system. Finally, the participants were given the SUS questionnaire. The participants were asked to rate the system with a scale of 1 as *strongly disagrees* to 5 as *strongly agree* based on the following questions:

Q1. I think that I would like to use this system frequently.

Q2. I found the system unnecessarily complex.

Q3. I thought the system was easy to use.

Q4. I think that I would need the support of a technical person to be able to use this system.

Q5. I found the various functions in this system were well integrated.

Q6. I thought there was too much inconsistency in this system.

Q7. I would imagine that most people would learn to use this system very quickly.

Q8. I found the system very cumbersome to use.

Q9. I felt very confident using the system.

Q10. I needed to learn a lot of things before I could get going with this system.

“To calculate the score, first we sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For the items 1,3,5,7 and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8 and 10 the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100” (Brooke, 1996). The results from the questionnaire about how useful of the STCMS are shown in figure 7.6. The results indicate that the participants found that: the use of STCMS is attractive, the system is easy to use and it provides the participants with related software testing terms and test cases.

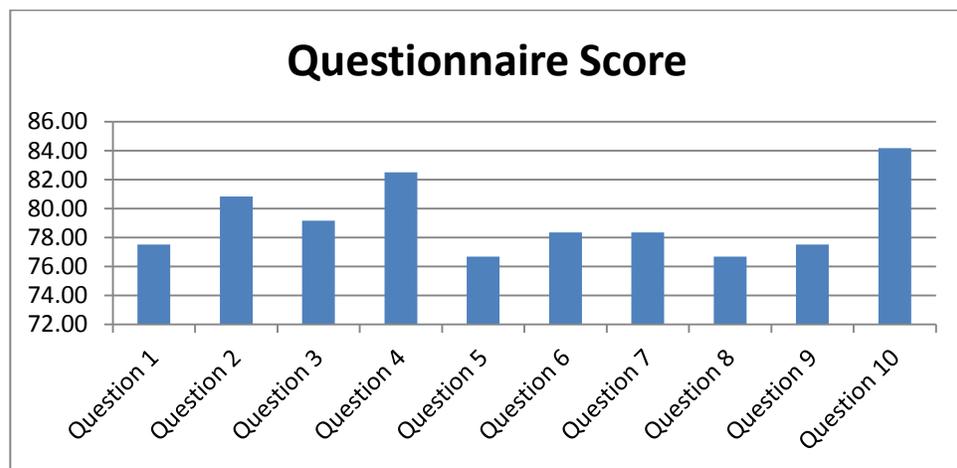


Figure 7-6 Questionnaire Results

As the score shows, users found STCMS significantly better suited to the required task. From a range of 0 to 100 , users gives the STCMS an average score of 79.17 (**Appendix D SUS DATA**).The interpretation of the scores describing the acceptability of the system is according of figure 7-7.this shows that the STCMS is EXCELLENT

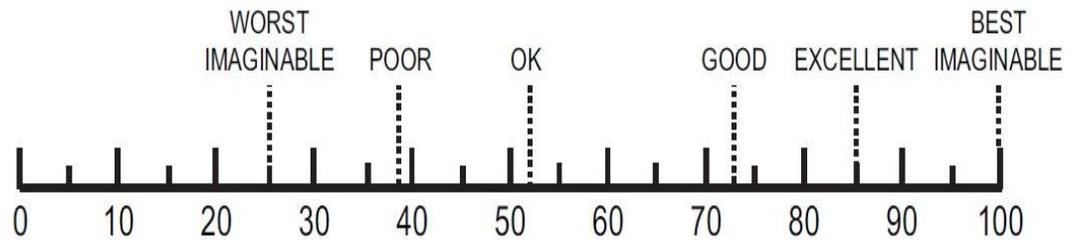


Figure 7-7 The Acceptability of SUS Score Adapted from (Bangor, Kortum, & Miller, 2008)

The STCMS retrieval information consists of well-structured test cases items and software artefact to aid tester in linking test cases to their sources. Our results show that this composition gained users satisfaction. Hence we conclude that utilizing of the semantic technology allows us to provide information with particular interest to the tester.

- II. Validity:** The validation process is necessary to ensure the trustworthiness of STCMS’ features and results. The classification’s schemas are selected based on tailoring to what have been usually used in Software Engineering. Table 7.2 illustrates the different aspects covered in the validation process via a checklist to control the constancy.

Table 7-2 Validation Checklist

Validity Criteria	Checklist	Result	Feature
Construct	Does the system design cover the objectives of the Case Study?	✓	
Internal	Do the system outputs reflect the objectives?	✓	
External	Have the beneficiaries of the system been identified?	✓	
Reliable	Has the data collection been standardized?	✓	
Functional	Do the test cases cover all functions of the system?		✓
Usable	Does it require users to learn any special programming languages?		✓
Scalable	Are there any limitations?		✓

From the above table, the checklist questions had been developed and answered throughout the research phases. This development is represented in three main points:

- a) We identified the objectives that reflect the research purpose, designed the case study accordingly, and ensured that the outputs present these objectives.
- b) We named the concerned groups to circulate the findings and generated a standard format to collect the data.
- c) We prepared the test cases (**Appendix B-3 STD**) to test each function in the system, ensured that the interfaces are friendly and that any extra coding or help is not needed, and acknowledged the functions and storage limitation of the system.

7.2.4 Performance of Semantic Search

The test case search feature is integrating the semantic technology as discussed in section 4.4. The feature, as shown in the GUI figure 7.8, helps users to hunt for the required test cases. The text field search converts the query into tokens. The tokens are then matched semantically with the Software Testing Ontology as shown in section 7.2.1. This helps users to identify more related search terms.

The screenshot displays the 'TestCases Semantic Search' web application. At the top, a navigation bar includes links for Home, Create TestCase, Search TestCase By ID, Search TestCase, View TestCases, Software Testing Ontology, and Resources. The main heading is 'TestCases Semantic Search'. Below the heading is a search input field containing the text 'Function' and a 'Search' button. To the left of the search results is a sidebar listing various ontology classes with their terms and owl:Class identifiers. The main content area shows 'Software Testing Ontology Results' with a 'Navigation Bar' set to 'Thing'. The results list includes 'Tester', 'Human', and 'Software_Tool'. Below this, the 'TestCases Results' section displays a list of 10 test cases, each with a URL and a 'Details' link.

Test Case ID	URL	Action
(1)	http://purl.org/dc/elements/1.1/#Logout	Details
(2)	http://purl.org/dc/elements/1.1/#QuickLinks	Details
(3)	http://purl.org/dc/elements/1.1/#Login	Details
(4)	http://purl.org/dc/elements/1.1/#CalculateHistogram	Details
(5)	http://purl.org/dc/elements/1.1/#iNormalizePattern	Details
(6)	http://purl.org/dc/elements/1.1/#iCount	Details
(7)	http://purl.org/dc/elements/1.1/#iCalculateProcessTime	Details
(8)	http://purl.org/dc/elements/1.1/#AccessHomepage	Details
(9)	http://purl.org/dc/elements/1.1/#Searchbyreferenceortracepoint	Details
(10)	http://purl.org/dc/elements/1.1/#iCheckVersion	Details

Figure 7-8 Test Case Semantic Search

To evaluate the accuracy and completeness of the search result, we used the precision and recall measurement adopted from (Chinchor & Sundheim, 1993). The precision is measured by the number of relevant documents from the total documents retrieved, while the recall is measured by the relevant documents retrieved from the total relevant documents that exist. The measurement for precision and recall using the following formula:

$$\text{Precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{documents retrieved}\}|}{|\{\text{documents retrieved}\}|}$$

$$\text{Recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{documents retrieved}\}|}{|\{\text{relevant documents}\}|}$$

Our STO covers four general classifications in software testing filed as described in IV of section 5.1. In order to test the test cases collected as described in section 6.1.2, we set four queries description to carry out the evaluation. The four classifications and queries are listed in table 7.3.

Table 7-3 Queries Vs General Classification

General Classification		Queries Description
Tester	>	Set of queries that extract Test Cases based on tester details, for instance creator name or group id
Task_Testing	>	Set of queries that extract Test Cases based on testing tasks, for instance subject of testing or testing type
Artefact	>	Set of queries that extract Test Cases based on linked artefact and testware, for instance source and relation
Environment	>	Set of queries that extract Test Cases based on the purpose, objective or input descriptions

Tables 7.4 to 7.7 show the precision and recall analysis results of the throughput of semantic test case search. The search data taken from testing scenarios developed while in the period of data collection. The first two tables (table 7.4 and table 7.5) show the Tester and Task scenarios whereas the last two tables (table 7.6 and table 7.7) show the Artefact and Environment scenario.

I. Tester Query

Table 7-4 Tester Search Terms Evaluation

#	Search Term	Relevant Test Cases	Retrieved Test Cases	Recall	Precision
1	Developers	21	21	100%	100%
2	Faduma	7	7	100%	100%
3	Mansoor	10	10	100%	100%
4	Zak	10	10	100%	100%

In the data collection we identified 4 different recourses as described in section 6.3.2. Hence, we tested the 4 possible scenarios for testers as shown in the Search Term column. Table 7.4 results show 100% in both precision and recall for all scenarios. Tester scenario is considered direct information that is known by the user. We expected these results as **Tester** is one of the main elements represented in the RDFS.

II. Task_Testing Query

Table 7-5 Task Testing Search Terms Evaluation

#	Search Term	Relevant Test Cases	Retrieved Test Cases	Recall	Precision
1	Functional	38	37	97%	100
2	Black Box	10	10	100%	100%

When filtering the collected test cases, we found them categorized into two types: Functional Testing and Black Box Testing. Therefore, we tested the two scenarios as shown in the Search Term column. Table 7.5 results show 100% for precision in both scenarios, whereas it was 97% to 100% in the recall. Although our expectation was to have 100% for both recall and precision, after analyzing the results, we observed that STCMS does not retrieve similar documents. We found that in the collected test cases, there were two similar test cases collected from different resources.

III. Artefact

Table 7-6 Artefact Search Terms Evaluation

#	Search Term	Relevant Test Cases	Retrieved Test Cases	Recall	Precision
1	iLogger	21	21	100%	100%
2	STCMS	10	10	100%	100%
3	MFIT	10	10	100%	100%
4	FWP	7	7	100%	100%

Standard development requires consistence in standard names for all software artefacts. For instance, the requirements of the iLogger system will start with the iLogger pretext, so do the designing and testing. STCMS links the Individual Test Case sources element to other software artefacts. STCMS was prototyped using the four main systems as described in section 6.3.2 and consequently, the testing considerate to test the four scenarios as shown in the Search Term columns. Table 7.6 results show 100% in both precision and recall for all scenarios.

IV. Environment

Table 7-7 Environment Search Terms Evaluation

#	Search Term	Relevant Test Cases	Retrieved Test Cases	Recall	Precision
1	File	13	15	100%	87
2	Home Page	19	20	100%	95%
3	Mobile	2	2	100%	100%
4	Server	10	12	100%	83%
5	Semantic	1	1	100%	100%
6	Personalize	5	4	80%	100%
7	Add	5	6	100%	83%

The collected test cases were created for different purposes. For the evaluation we randomly selected search terms that cover all aspects. Table 7.7 shows results between 80% to 100% for precision and recall.

7.3 Discussion

7.3.1 STCMS vs. Other Web Test Case Management System:

The main features of STCMS are represented in figure 7.9. For instance: Comprehensive Test Case Management comprises create, edit, view of the test case; RDFS representation, which has been discussed in the design sub-heading; Semantic Web Environment, which applies the Semantic Technology layers; and finally, Semantic Search, which relies on the STO.

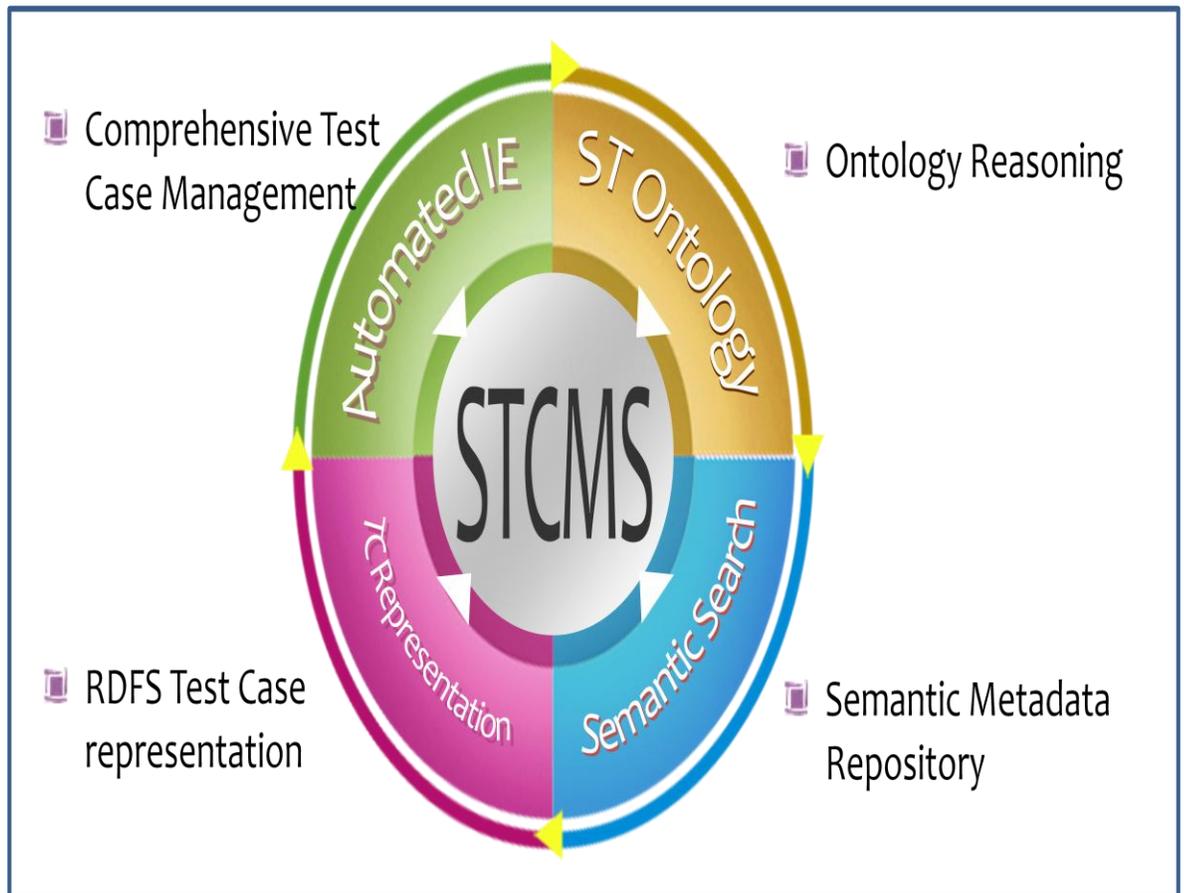


Figure 7-9 STCMS' main features

The comparisons of the innovative services of STCMS with the other Testing Management System tools are illustrated in Table 7.8. For instance, Search Approach feature in STCMS is based on the RDFS representation. The benefit of RDFS is to provide basic vocabulary for describing the hierarchies of test cases metadata and attributes, and specifying properties and relations among them.

Moreover, the mechanism of STCMS Storing feature is to store the test case annotation in relational database (MySQL), which increases the retrieval phase using the query language (SPARQL). Based on the mapping of the query words with the Software Testing explicit conceptual description (STO), the (Semantic Web) vision of supporting automate tasks and enabling agents to automatically discover the services to be fulfilled

Table 7-8 STCMS Vs Other Testing Tools

Tools	Type	Search Approach	Information Retrieval	Storing
ApTest Manager	Web based test management	Not applicable	Not applicable	Keyword Index
Chrysilla Test Case	Web-based service	Not applicable	Not applicable	Keyword Index
TestUP	Web based test management	Keyword Search	Not applicable	Keyword Index
STCMS	Semantic Case Management	Test	Semantic Discovery base on RDFS	Automatic Based on Software Testing Ontology Semantic Index

7.3.2 Benefit of using Semantic Technology:

The current web uses a human understandable format to display its content and services. The vision of the Semantic Web (considered as future web) aims to use a human and machine understandable format by data integration. In figure 7.10 and Figure 7.11, we illustrate the differences on how human and machine can access the test case in STCMS.

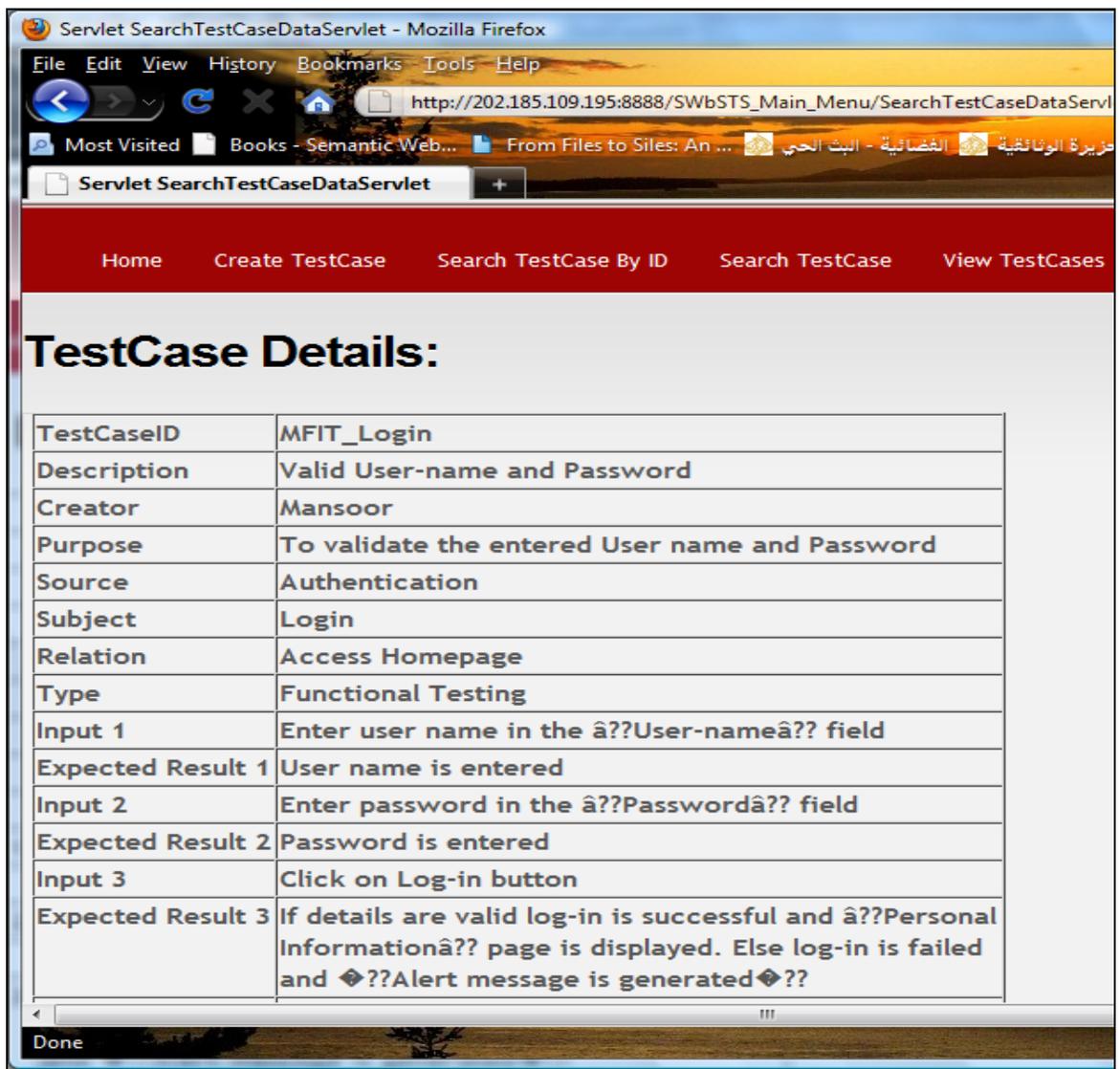


Figure 7-10 The Test Case seen by a human

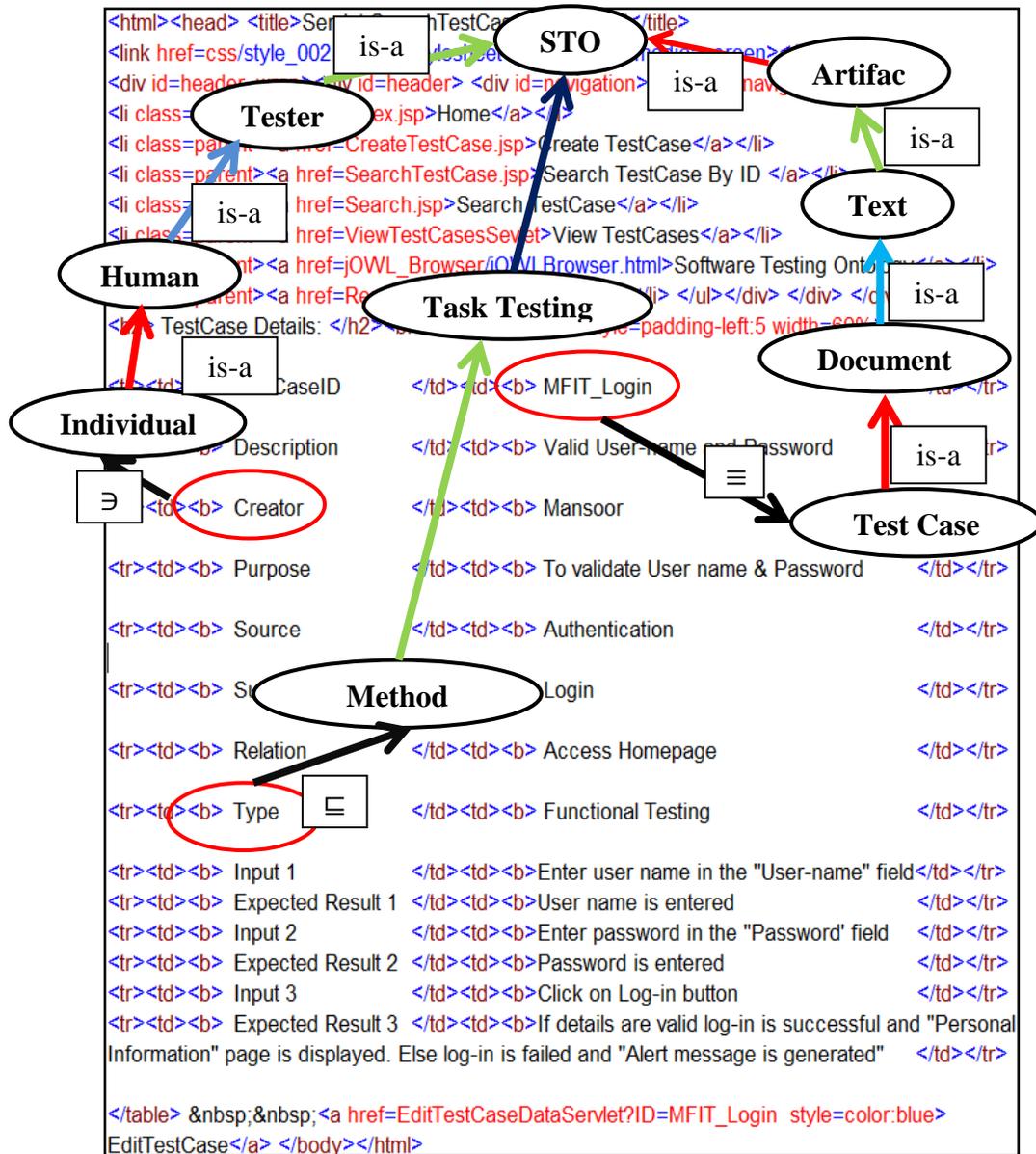


Figure 7-11 The Test Case seen by a machine

The goal here (Type is sub class of \sqsubseteq Method, Creator is instance \exists of Individual, and MFIT_Login is equivalent value \equiv of Test Case) is to show how machine can recognise the different information of test case and reason their relations. With these results, the vision of STCMS to automate tasks is achievable.

7.4 Summary

In this thesis, we described the benefits of representing individual test case and integrating conceptually connected testing terms. The results presented here show their significance by properly storing and utilizing the individual test cases, so they are easily found during future searches. This makes the testing process and management well-organized.

The results showed four main aspects: the incorporation of Software Testing Ontology, automated information extraction, representation of well-structured test cases, and semantic searches. We evaluated the **STO** using FaCT++ and Pellet plug-in reasoners provided by Protégé. This evaluation gave the STO accuracy with its DL-syntax for reasoning purposes. The **automated information extraction** evaluation matched semantic similarities between the retrieved concepts with the query concept, so the tester obtained the correct terms from all possibilities. We utilized the Login Test Case to evaluate the **representation of Well-Structured Test Cases**, through which we illustrated the benefits of a machine-readable representation. Finally, by using the precision and recall equation, we proved the efficiency of the semantic search mechanism. In conclusion, the STCMS is a unique product that stands above the rest.

8.0 Conclusion

The invention of a fully automated Software Testing System can be sometime away, yet our work is a step towards that destination. The challenge, though, is that the testing process is time-consuming and costly throughout production due to millions of individual test cases that are underutilized and mismanaged by testers that are unaccustomed to general asset management. However, in this piece of research-work, we showed a high expectation emerging from the significant results of integrating semantic technology with the test case management process to help software engineers to produce higher-quality software in a time effective manner at a lower cost.

To encapsulate, in this thesis we have discussed four main objectives, as stated below:

- ✓ **Objective 1:** To analyse and derive individual **Well-Structured Test Case** using RDFS
- ✓ **Objective 2:** To formalize terms for **Software Testing Ontology** and use the Ontology Web Language to represent it in such a way that it can easily be used by other automated tools, software agents and knowledge management
- ✓ **Objective 3:** To apply the Well-Structured Test Case representation, integrated with the Software Testing Ontology, to a **semantic information retrieval mechanism** to act as a *knowledge base system* for retrieving and managing knowledge in the domain of Software Testing
- ✓ **Objective 4:** To evaluate the approach in a **Semantic Management Application;** under the name Semantic Test Case Management System

To achieve our goal, we came out with questions that have been answered in the previous chapters. In Table 8.1, we illustrate where in this thesis these questions have been clarified and answered.

Table 8-1 Sections map showing where in thesis research questions answered

Objective	Questions	Chapters	Sections
<i>Objective 1</i>	<i>Q1. What do we understand about the weaknesses of the current testing – automation and management?</i>	2/4	2.3-4.1
<i>Objective 1</i>	<i>Q2. What is the value of individual test cases? Is there any need for a test case to be well-structured and represented individually? and what type of metadata and attributes need to be considered?</i>	2/4/6	2.2-4.2-6.2
<i>Objective 2</i>	<i>Q3. How to formulate well-known standard software testing terms in ontology to minimize the confusion that occurs among software testing practitioners?</i>	3/4/5	3.2-4.3-5.1-5.2
<i>Objective 3</i>	<i>Q4. How can we use the semantic technology for individual test case management to minimize the painstaking effort and time spent on auditing all test artefacts?</i>	3/4/6	3.1-4.4-6.4
<i>Objective 4</i>	<i>Q5. How to evaluate the TCMS efficiency and the reasoning of the formulated terms?</i>	7	7.1-7.2-7.3-7.4

Through answering these questions, we have come to certain conclusions and findings. From which, we proposed an effective contribution in the testing process by integrating semantic technology. The findings and contributions are discussed in the following sections.

8.1 Findings

Researchers around the globe are currently discussing automating the software testing process and the challenges of successfully producing such a system. From our investigation, we found a management problem, which if solved would definitely help in automating the testing process. We obtained the following findings, which reveal the weaknesses and potential solutions to managing and automating the current testing process:

- I. **Individual Test Cases:** Research reveals that the power of individual test cases, although playing a very crucial role in the test process, has been virtually ignored due to several factors.
 - A lack of quality management overseeing the usage of individual cases.
 - Individual test cases are not being reused efficiently due to poor organization and no link between the test cases and other test-ware and artefacts.

Our solution is to present these test cases in a well-structured semantic technology management, which we call STCMS. Test cases represented in our STCMS linked the individual test cases with other test-ware and artefacts, creating a real-time automated format for information retrieval. This linkage helps in making the

decisions on which test cases can be reused in various environments and cases based on the available information.

II. Software Testing Concepts and Terms: Software-testing practitioners often interpret similar terms in different ways causing misunderstandings and confusion, which has resulted in delays within the testing process as repeatedly demonstrated. This misunderstanding and confusion affects the management of the testing process because each party (i.e. testers, managers, share holders) identifies a different component by the term used instead of referring to the same concept, as it does. So, we propose that the solution lies in **ontologies**. These are artefacts in knowledge-based systems, which define concepts and terms, streamlining them into a single meaning. When the misunderstandings and confusion are eliminated, the specific relation and meaning of the domain structure are exposed, and the process is then simplified so it can be well managed.

III. Test Case Management & Search: On a daily basis, testers always create vast amount of test cases to test any software product. Current TCMS are using a relational database, which stores isolated test cases. In order to utilize these stored individual test cases, we are led to search through this un-semantic database using the normal “keyword search” approach. This is an ineffective method to find all of the applicable test cases available. When we **implemented** our Semantic Test Case Management System (STCMS), we attached individual test cases with other testing artefacts semantically. We found the semantic search to be a useful search to link reusable common-share knowledge among test cases and testing practitioners. Not only does this create reusability of test cases, but also aiding the practitioners with additional concepts

related to their search term may expound their searching functionality beyond their initial search.

8.2 Contribution

We focused our contribution to advance the testing committee knowledge on the following points:

- I. Automated Information Extraction:** Integrating semantic technology in a TCMS contributes to automated support for retrieving, storing and tracing any individual test cases stored in the system.

- II. Representing well-structured Individual Test Cases:** A Well-Structured Test Case in RDFS form input into our STCMS reflects how test cases stored in semantic format can be easily retrieved for reuse in a future test case. The representative illustrates the power of the individual test cases that are “tagged” for knowledge-based semantic searches in order for repeated recognition.

- III. Software Test Ontology:** Supporting the testing process with the Software Testing Ontology captures the logical relationship between standard software testing terms. These streamlined definitions in knowledge-based systems provide various terms with a structured meaning for the testing process.

- IV. Semantic Search:** This approach changes the search capability in the testing domain knowledge. By providing the testers and practitioners with a variety of information related to their search, we make the search experience more beneficial.

8.3 Future Work

Our desire for this research topic is limitless, and this thesis is just the beginning. Possible extensions that we are looking at in the current work can be summarized in the following questions:

1. How to represent other test artefact, in particular Use Case using the Semantic Technology?
2. How to match between the represented test case and represented Use Case for reasoning and test cases auto extraction?
3. What is the benefit of having Semantic Agent as a main component for the Semantic Software Testing System?

Finally in this chapter, we have shown how the objectives of this thesis have been achieved and where the research questions have been answered. Moreover, we managed to summarise the findings from answering the derived questions, the contributions made from our proposed solutions and glance of ideas for future research work.

References

- Ammann, P., & Offutt, J. (2008). *Introduction to software testing*: Cambridge Univ Pr.
- Antoniou, G., & Harmelen, F. v. (2008). *A Semantic Web Primer*: The MIT Press.
- Antoniou, G., & Harmelen, F. v. (2009). Web Ontology Language: OWL. In S. Staab & R. Studer (Eds.), *Handbook on Ontologies* (pp. 91-110): Springer Berlin Heidelberg.
- Arpírez, J., Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2003). WebODE in a nutshell. *Ai Magazine*, 24(3), 37.
- Bai, X., Lee, S., Tsai, W. T., & Chen, Y. (2008). *Ontology-based test modeling and partition testing of web services*. Paper presented at the IEEE International Conference on Web Services, 2008. ICWS'08. , Beijing.
- Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6), 574-594. doi: 10.1080/10447310802205776
- Barbosa, E. F., Nakagawa, E. Y., & Maldonado, J. C. (2006). *Towards the establishment of an ontology of software testing*. Paper presented at the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006).
- Bechhofer, S., Horrocks, I., Goble, C., & Stevens, R. (2001). OilEd: a reason-able ontology editor for the semantic web. *KI 2001: Advances in Artificial Intelligence*, 396-408.
- Beizer, B. (2002). *Software testing techniques*: Dreamtech Press.
- Berners-Lee, T. (January 6, 1997). Axioms of Web Architecture: Metadata Retrieved 15/12, 2010, from <http://www.w3.org/DesignIssues/Metadata.html>
- Bertolino, A. (2007). *Software Testing Research: Achievements, Challenges, Dreams*.

- Brachman, R. J., & Levesque, H. J. (2004). *Knowledge representation and reasoning*: Morgan Kaufmann Pub.
- Breitman, K. K., Casanova, M. A., & Truszkowski, W. (2007). *Semantic Web: concepts, technologies and applications*: Springer Verlag.
- Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189, 194.
- Chinchor, N., & Sundheim, B. (1993). *MUC-5 evaluation metrics*. Paper presented at the Proceedings of the 5th conference on Message understanding, Baltimore, Maryland.
- Chunyue, L. (2011). Test Automation Method for Software Programs (Vol. US 7930683B2). United States: SAP AG, Walldorf (DE).
- Core, D. (1995). Dublin Core Metadata Element Set, Version 1.1. from <http://dublincore.org/documents/dces/>
- Damm, L.-O., Lundberg, L., & Olsson, D. (2005). Introducing Test Automation and Test-Driven Development: An Experience Report. *Electronic Notes in Theoretical Computer Science*, 116(0), 3-15. doi: 10.1016/j.entcs.2004.02.090
- Dengel, A. (2007). Knowledge Technologies for the Social Semantic Desktop. In Z. Zhang & J. Siekmann (Eds.), *Knowledge Science, Engineering and Management* (Vol. 4798, pp. 2-9): Springer Berlin / Heidelberg.
- Desai, H. D. (1994). *Test Case Management System (TCMS)*. Paper presented at the Global Telecommunications Conference, 1994. GLOBECOM '94. Communications: The Global Bridge., IEEE.
- Domingue, J. (1998). Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web.

- Farquhar, A., Fikes, R., & Rice, J. (1997). The ontolingua server: A tool for collaborative ontology construction. *International Journal of Human-Computers Studies*, 46(6), 707-727.
- Fraser, G., & Zeller, A. (2011). *Exploiting common object usage in test case generation*. Paper presented at the IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST) Berlin, Germany.
- Fu, R.-x., Yue, X., Song, M., & Xin, Z.-h. (2008). An architecture of knowledge management system based on agent and ontology. *The Journal of China Universities of Posts and Telecommunications*, 15(4), 126-130.
- García-Sánchez, F., Valencia-García, R., Martínez-Béjar, R., & Fernández-Breis, J. T. (2009). An ontology, intelligent agent-based framework for the provision of semantic web services. *Expert Systems with Applications*, 36(2, Part 2), 3167-3187. doi: DOI: 10.1016/j.eswa.2008.01.037
- Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*: Springer Verlag.
- Grobe, M. (2009). *RDF, Jena, SparQL and the 'Semantic Web'*. Paper presented at the Proceedings of the 37th annual ACM SIGUCCS fall conference, St. Louis, Missouri, USA.
- Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2), 199-220. doi: 10.1006/knac.1993.1008
- Gupta, P., Malik, S. K., Prakash, N., Rizvi, S. A. M., & Arora, M. (2004). RDF: AN ANALYSIS. *Science (Vol. I)*, 34513, 0034515.

- Gupta, P., & Surve, P. (2011). *Model based approach to assist test case creation, execution, and maintenance for test automation*. Paper presented at the Proceedings of the First International Workshop on End-to-End Test Script Engineering.
- Hall, W., & O'Hara, K. (2009). Semantic Web. *Robert A. Meyers (ed.), Encyclopedia of Complexity and Systems Science*.
- Han, L., Feng, C., Hongji, Y., He, G., Chu, W. C. C., & Yuansheng, Y. (2009, 20-24 July 2009). *An Ontology-Based Approach for GUI Testing*. Paper presented at the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09), Seattle, Washington, USA.
- Happel, H. J., & Seedorf, S. (2006). *Applications of ontologies in software engineering*. Paper presented at the Proc. of Workshop on Semantic Web Enabled Software Engineering"(SWESE) on the ISWC.
- Hayes, L. (2000). Establishing a Test Automation Function. *Journal of Software Testing Professionals*.
- Heiser, J. E. (1997). *An Overview of Software Testing*. Paper presented at the IEEE Autotestcon Proceedings (AUTOTESTCON, 97).
- Heiskanen, H., Maunumaa, M., & Katara, M. (2012). A Test Process Improvement Model for Automated Test Generation. *Product-Focused Software Process Improvement*, 17-31.
- Hendler, J. (2001). Agents and the semantic web. *IEEE Intelligent Systems*, 16(2), 30-37.
- Hendler, J. (2010). Web 3.0: The Dawn of Semantic Search. *Computer*, 43(1), 77-80.
- Hong, Z. (2006, 17-21 Sept. 2006). *A Framework for Service-Oriented Testing of Web Services*. Paper presented at the COMPSAC '06. 30th Annual International Computer Software and Applications Conference, 2006. , Chicago, USA.

- Horrocks, I. (2002). DAML+ OIL: a description logic for the semantic web. *Bulletin of the Technical Committee on*, 51(4).
- Husain, M., McGlothlin, J., Masud, M. M., Khan, L., & Thuraisingham, B. M. (2011). Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. *Knowledge and Data Engineering, IEEE Transactions on*, 23(9), 1312-1327.
- IFLA. (October 24, 2005). Digital Libraries: Metadata Resources. Retrieved 15/12, 2010, from <http://archive.ifla.org/II/metadata.htm>
- ISTQB. (2002). International Software Testing Qualifications Board. Retrieved February 03, 2011, from <http://istqb.org/display/ISTQB/Home>
- John Davies, Rudi Studer, & Warren, P. (2006). *Semantic Web Technologies, trends and research in ontology-based systems*: Wiley.
- Jorgensen, P. C. (2008). *Software testing: a craftsman's approach*: Auerbach Publications.
- Juan, Z., Lizhi, C., Weiqing, T., Zhenyu, L., & Ying, L. (2009). *A Dynamic Metrics Method for Test Case Reuse Based on Bayesian Network*. Paper presented at the CiSE 2009. International Conference on Computational Intelligence and Software Engineering, 2009. , Wuhan, China.
- Kamde, P. M., Nandavadekar, V. D., & Pawar, R. G. (2006, 21-23 June 2006). *Value of Test Cases in Software Testing*. Paper presented at the IEEE International Conference on Management of Innovation and Technology, Singapore.
- Kaner, C. (2003). What is a good test case. *Relation*, 10(1.100), 5569.
- Kang, S. H., & Lau, S. K. (2007). *Ontology Revision on the Semantic Web: Integration of belief revision theory*. Paper presented at the 40th Annual Hawaii International Conference on System Sciences (HICSS), Hawaii.

- Karp, P., Chaudhri, V., & Thomere, J. (1999). XOL: An XML-based ontology exchange language. *Version 0.3, July, 3*.
- Kekkonen, T., Kanstrén, T., & Heikkinen, J. (2012). *Experiences in Test Automation for Multi-Client System with Social Media Backend*. Paper presented at the VALID 2012, The Fourth International Conference on Advances in System Testing and Validation Lifecycle.
- Lack of Test Case Management Threatens Software Quality. (26 June 2008). *Business Wire*.
- Li, B. M., Xie, S. Q., & Xu, X. (2011). Recent development of knowledge-based systems, methods and tools for One-of-a-Kind Production. *Knowledge-Based Systems, 24(7)*, 1108-1119. doi: DOI 10.1016/j.knosys.2011.05.005
- Lin, D. (1998). *An information-theoretic definition of similarity*.
- Louridas, P. (2011). Test Management. *Software, IEEE, 28(5)*, 86-91.
- Luke S, H. J. (2000). SHOE 1.01. Proposed Specification. Technical Report. Retrieved 14th November, 2010, from <http://www.cs.umd.edu/projects/plus/SHOE/>
- Mahadevan, G. (2012). Semantic Information and Web based Product Recommendation System—A Novel Approach. *International Journal of Computer Applications, 55(9)*, 10-14.
- Majchrzak, T. A. (2010). *Best practices for technical aspects of software testing in enterprises*. Paper presented at the International Conference on Information Society (i-Society), London, United Kingdom.
- Mary Jean, H. (2000). *Testing: a roadmap*. Paper presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland.
- McGuinness, D., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation, 10*, 2004-2003.

- Miller, K., & Voas, J. (2006). Software test cases: is one ever enough? *IT Professional*, 8(1), 44-48.
- Mordechai, B.-M. (2008). Towards management of software as assets: A literature review with additional sources. *Information and Software Technology*, 50(4), 241-258. doi: 10.1016/j.infsof.2007.08.001
- Myers, G. J. (2004). *The art of software testing* (2nd ed.): Wiley.
- Nakagawa, E. Y., Simao, A., Ferrari, F., & Maldonado, J. C. (2007). *Towards a reference architecture for software testing tools*. Paper presented at the SEKE.
- Natasha, F. N., & Deborah, M. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology* (K. S. Laboratory, Trans.): Stanford University.
- Noy, N., Fergerson, R., & Musen, M. (2000). The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In R. Dieng & O. Corby (Eds.), *Knowledge Engineering and Knowledge Management Methods, Models, and Tools* (Vol. 1937, pp. 69-82): Springer Berlin / Heidelberg.
- Osman, T., Thakker, D., Schaefer, G., Leroy, M., & Fournier, A. (2007). *Semantic Annotation and Retrieval of Image Collections*. Paper presented at the Proceedings 21st European Conference on Modelling and Simulation.
- Patton, R. (2001). *Software testing*: Sams.
- Perez, A., Angele, J., Lopez, M., Christophides, V., Stutt, A., & Sure, Y. (2002). A survey on ontology tools. *Citeseer*, .
- Puri, A. (2012). Automation Framework of Browser Based Testing Tool Watir. *International Journal of Computers & Distributed System (IJCDS)*, 1(3), 113-117.
- Rafi, D. M., Moses, K. R. K., Petersen, K., & Mantyla, M. (2012). *Benefits and limitations of automated software testing: Systematic literature review and practitioner survey*.

Paper presented at the 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland.

- Rathod, M. D., Prajapati, M. R., & Singh, M. A. (2012). Applying Semantic Web Mining Technologies In Personalized E-Learning. *International Journal of Engineering, 1*(3).
- Rex, B. (2002). *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing* (2nd ed.): Robert Ipsen.
- Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), 131-164. doi: 10.1007/s10664-008-9102-8
- Serhatli, M., & Alpaslan, F. N. (2009). An ontology based question answering system on software test document domain. *World Academy of Science*.
- Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The semantic web revisited. *IEEE Intelligent Systems, 21*(3), 96-101.
- Simperl, E., Mochol, M., & Bürger, T. (2010). Achieving maturity: The state of practice in ontology engineering in 2009. *International Journal of Computer Science and Applications, 7*(1), 45-65.
- Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web, 5*(2), 51-53.
- Strobbe, C., Herramhof, S., Vlachogiannis, E., & Velasco, C. (2006). Test Case Description Language (TCDL): Test Case Metadata for Conformance Evaluation. In K. Miesenberger, J. Klaus, W. Zagler & A. Karshmer (Eds.), *Computers Helping People with Special Needs* (Vol. 4061, pp. 164-171): Springer Berlin / Heidelberg.

- Strobbe, C., & Velasco, C. (2005). Test Suites' State of the Art and Quality Assurance Methods for W3C Recommendations (pp. 45). BenToWeb.
- Su, X., & Ilebrikke, L. (2006). A Comparative Study of Ontology Languages and Tools. In A. Pidduck, M. Ozsu, J. Mylopoulos & C. Woo (Eds.), *Advanced Information Systems Engineering* (Vol. 2348, pp. 761-765): Springer Berlin / Heidelberg.
- Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., & Wenke, D. (2002). OntoEdit: Collaborative ontology development for the semantic web. *The Semantic Web—ISWC 2002*, 221-235.
- Tauhida, P., Scott, T., & George, G. (2007). *A case study in test management*. Paper presented at the Proceedings of the 45th annual southeast regional conference, Winston-Salem, North Carolina.
- Terminology, I. S. G. o. S. E. (1990). *IEEE Std 610.12-1990*, 1.
- Tim, B.-L., James, H., & Ora, L. (2001). The Semantic Web. *Scientific American Magazine*.
- Tonta, Y. (2011). Analysis of Search Failures in Document Retrieval Systems: A Review. *Public Access-Computer Systems Review*, 3(1).
- Veenendaal, E. (2010). Standard glossary of terms used in Software Testing, Version 2.1. *International Software Testing Qualification Board*.
- W3C. Test Metadata. *Test Metadata*. 2011, from <http://www.w3.org/TR/2005/NOTE-test-metadata-20050914/>
- Weibel, S., Kunze, J., Lagoze, C., & Wolf, M. (1998). Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*, 2413.
- Whittaker, J. A. (2000). What is software testing? And why is it so hard? *IEEE software*, 17(1), 70-79.

- Wiklund, K., Eldh, S., Sundmark, D., & Lundqvist, K. (2012). *Technical Debt in Test Automation*. Paper presented at the IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), Montréal.
- Woodward, M. R., & Hennell, M. A. (2004). Strategic benefits of software test management: a case study. *Journal of Engineering and Technology Management*, 22(1-2), 113-140. doi: 10.1016/j.jengtecman.2004.11.006
- Yahaya, N. (2008). *Using Semantic Web Languages in Representing Test Cases*. Paper presented at the Information Technology and Multimedia, UNITEN, Malaysia.
- Yıldırım, H., Chaoji, V., & Zaki, M. J. (2012). GRAIL: a scalable index for reachability queries in very large graphs. *The VLDB Journal*, 21(4), 509-534.
- Yuan, G. (2011). *Study of Implementation of Software Test Management System based on Web*. Paper presented at the IEEE 3rd International Conference on Communication Software and Networks (ICCSN), Beijing.
- Zhao, P., Di, L., Yu, G., Yue, P., Wei, Y., & Yang, W. (2009). Semantic Web-based geospatial knowledge transformation. *Computers & Geosciences*, 35(4), 798-808. doi: DOI: 10.1016/j.cageo.2008.03.013

Appendixes

Appendix A: Ontology Vocabulary

Appendix A-1: STO Terms Classification

Terms	Definition	Terms	Definition
Tester	The individual, Team or software conduct the test	Human	Living thing
Context	The circumstance to perform the testing task	Software_Tool	Code of Application
		Scope	Testing activities occurs in various development stages
Activity	The primary, organizational or supporting activities of testing task	Purpose	Testing activities occurs for various purposes
		Intrinsic	Activities occurs within the system
Method	The way of performing the testing task	Extrinsic	Activities occurs without the system
		Approach	Approaches to perform testing task
		Practice	Practices to perform testing task
Artefact	Anything Tester-made such as (text, image, or standard) related to the testing task	Technique	Techniques to perform testing task
		Text	String of character
		Images	Picture or chart
Environment	The aggregate surrounding such as (Software, Hardware, Features) of Testing	Standard	Approved documents or guide distinguishing characteristic of software and software testing
		Hardware	Device integrated with Software Application Program
		Software	Application

Appendix A-1: STO Terms Classification

Tester	
Human	Software_ Tool
Assessor	Analyzer
	Automated_ Testware
	Bug_ Tracking_ Tool
	Balanced_ Scorecard
Change_ Control_ Board Checker	Capture-Playback_ Tool
Configuration_ Control_ Board (Ccb)	Capture-Replay_ Tool
	Code_ Analyzer
	Comparator
	Configuration_ Management_ Tool
	Coverage_ Measurement_ Tool
	Coverage_ Tool
	Debugger
	Debugging_ Tool
	Defect_ Management_ Tool
	Defect_ Tracking_ Tool
	Driver
	Dynamic_ Analysis_ Tool
	Error_ Seeding_ Tool
	Fault_ Seeding_ Tool
	Hyperlink_ Test_ Tool
Inspection_ Leader	Incident_ Management_ Tool
Inspector	Instrumenter
Lead_ Assessor	Load_ Testing_ Tool
Moderator	Modelling_ Tool
	Monitor
	Monitoring_ Tool
Pair_ Testing	Performance_ Testing_ Tool
	Program_ Instrumenter
Recorder	Record-Playback_ Tool
Reviewer	Requirements_ Management_ Tool
	Review_ Tool
Scribe	Security_ Testing_ Tool
	Security_ Tool
	Static_ Analysis_ Tool
	Static_ Analyzer
	Static_ Code_ Analyzer
	Stress_ Testing_ Tool
Test_ Leader	Test_ Comparator
Test_ Manager	Test_ Data_ Preparation_ Tool
Test_ Process_ Group	Test_ Design_ Tool
Test_ Process_ Improver	Test_ Driver

Appendix A-1: STO Terms Classification

Test_Execution_Tool
Test_Generator
Test_Management_Tool
Test_Tool
Unit_Test_Framework

Appendix A-1: STO Terms Classification

Environment		
Features	Hardware	Software
Accuracy		
Adaptability		
Analyzability		
Anomaly		
Attractiveness		
Availability		
Behavior		Bespoke_ Software
Bug		Buffer
Buffer_ Overflow		
Changeability		Commercial_ Off-The- Shelf_ Software
Co-Existence		Compiler
Compliance		Component
Complexity		Cots
		Custom_ Software
Defect		
Deviation		
Efficiency	Emulator	
Error		
Error_ Tolerance		
Executable		
Exercised		
Fail		
Failure		
Fault_ Tolerance		
Functionality		
Incident		Installation_ Wizard
Installability		
Interoperability		
Learnability		
Maintainability		Module
Maturity		
Memory_ Leak		
Milestone		
Mistake		
Non-Conformity		
Operability	Operational_ Environment	Off-The-Shelf_ Software Operational_ Environment Oracle
Pass		
Performance		
Portability		

Appendix A-1: STO Terms Classification

Priority		
Probe_Effect		
Problem		
Product_Risk		
Project_Risk		
Quality		
Recoverability		Resource_Utilization
Reliability		
Replaceability		
Robustness		
Safety	Simulator	Safety_Critical_System
Scalability	Storage	Scripting_Language
Software_Test_Incident	Stub	Standard_Software
Stability		System
State_Transition		System_Of_Systems
Suitability		
Test_Execution_Phase	Test_Harness	Test_Bed
Test_Fail		Test_Environment
Test_Incident		Test_Rig
Test_Pass		
Testability		
Test_Session		
Time_Behavior		
Traceability		
Understandability		
Usability		

Appendix A-1: STO Terms Classification

Artefact				
Text				
Code	Document	Data	Measurement	
	Abstract_ Test_ Case			Actual_ Outcome Actual_ Result Agile_ Manifesto Audit_ Trail
Basic_ Block	Basis_ Test_ Set	Boundary_ Value_		Boundary_ Value
Branch	Blocked_ Test_ Case	Coverage		
Branch_ Condition		Branch_ Condition_ Combination_ Coverage Branch_ Condition_ Coverage Branch_ Coverage		
Compound_ Condition	Cause-effect_ Decision_ Table Charter Concrete_ Test_ Case	CASE CAST Chow's_ Coverage_ Metrics Condition_ Combination_ Coverage Condition_ Coverage Condition_ Coverage Cost_ Of_ Quality Critical_ Success_ Factor		Classification_ Tree Component_ Specification Configuration Control_ Flow Control_ Flow_ Path Corporate_ Dashboard Coverage_ Item
Dead_ Code	Decision_ Table Deliverable	Data_ Flow_ Coverage Decision_ Condition_ Coverage Decision_ Coverage Defect_ Density Defect_ Detection_ Percentage (DDP) Domain		Data_ Definition Decision_ Outcome Definition-use_ Pair
Entry_ Point		Equivalence_ Partition_ Coverage		Exit_ Point Expected_ Outcome Expected_ Result

Appendix A-1: STO Terms Classification

		Fault_Density Fault_Detection_Percentage (FDP)	False-fail_Result False-pass_Result False-positive_Result False-negative_Result Feasible_Path
Hyperlink	High_Level_Test_Case		
	Installation_Guide	Input_Domain	Indicator Infeasible_Path Input Input_Value
LCSAJ	Load_Profile Logical_Test_Case Low_Level_Test_Case	LCSAJ_Coverage	
Multiple_Condition		Multiple_Condition_Coverage Modified_Condition_Decision_Coverage Modified_Multiple_Condition_Coverage N-switch_Coverage	Maturity_Level Mean_Time_Between_Failures Mean_Time_To_Repair
Orthogonal_Array	Operational_Profile	Output_Domain	Outcome Output Output_Value
	Performance_Profiling	Path_Coverage	Path Postcondition Precondition Predicted_Outcome Pseudo-random
	Reliability_Growth_Model	Risk_Category Risk_Type	Result
Source_Statement	State_Table	Statement_	Specified_Input

Appendix A-1: STO Terms Classification

Artefact				
Standard				
Guide	Criteria	Report	Plan	Term
Agile_ Software_ Development	Acceptance_ Criteria	Assessment_ Report		Baseline Benchmark _Test Best_ Practice
Capability_ Maturity_ Model (CMM) Capability_ Maturity_ Model_ Integration (CMMI) Content- based_ Model Cyclomatic_ Complexity Cyclomatic_ Number	Completion_ Criteria			Certification Code Configuration _Item Continuous_ Representatio n
Deming_ Cycle				Data_ Flow Dashboard
European_ Foundation_ for_ Quality_ Management	Entry_ Criteria Exit_ Criteria			Emotional_ Intelligence
				Failure_ Mode Failure_ Rate Frozen_ Test_ Basis Functional_ Requirement
IDEAL Acting Diagnosin g Establishi ng Initiating		Incident_ Report Item_ Transmittal_ Report		

Appendix A-1: STO Terms Classification

Learning				Key_ Performance_ Indicator
		Level_ Test_ Plan		Lifecycle_ Model
		Master_ Test_ Plan		Maturity_ Model Measure Measurement_ Scale Metric Manufacturing- based_ Quality
				Non-functional_ Requirement
Process_ Model	Pass-Fail_ Criteria	Phase_ Test_ Plan Project_ Test_ Plan		Pareto_ Analysis Performance_ Indicator Pointer Process Process_ Assessment Process_ Improvement Product-based_ Quality Project Project_ Retrospective Qualification Quality Quality_ Attribute Quality_ Characteristic Quality_ Gate Requirement Requirements_ Phase Release_ Note
Rational_ Unified_ Process Root_ Cause				

Appendix A-1: STO Terms Classification

SCRUM	Suspension_	Software_		Scorecard
Software_	Criteria	Test_		Software_
Process_		Incident_		Life_Cycle
Improvement		Report		Software_
Staged_				Product_
Representatio				Characteristic
n				Software_
				Quality_
				Characteristic
				Status_
				Accounting
				Specification
Test_Basis	Test_	Test_	Test_Plan	Test
Test_Case	Completion_	Incident_		Test_
Test_Case_	Criteria	Report		Automation
Specification		Test_		Test_Cycle
Test_		Improvement		Test_Level
Design_		_ Plan		Test_Log
Specification		Test_Item_		Test_Oracle
Test_		Transmittal_		Test_
Maturity_		Report		Procedure
Model		Test_Report		Test_
(TMM)		Test_		Procedure_
Test_		Summary_		Specification
Maturity_		Report		Test_Record
Model_				Test_Run_
Integrated				Log
(TMMi)				Test_
				Scenario
				Test_Type
				Testability_
				Review
				Testware
				Transcendent-
				based_
				Quality
				Total_
				Quality_
				Management
				Transactional
				_ Analysis
				Work_
				Breakdown_
				Structure
				User-based_
				Quality
V-model				

Artefact
Image
Call_ Graph
Cause-effect_ Diagram
Cause-effect_ Graph
Control_ Flow_ Graph
Fishbone_ Diagram
State_ Diagram
Ishikaw_ Diagram
Mind-Map

Appendix A-1: STO Terms Classification

Context.	
Purpose	Scope
Acceptance	Agile Testing
Acceptance_Testing	Ad Hoc Testing
Accuracy_Testing	
Alpha_Testing	
Beta_Testing	Big-Bang_Testing
Black-Box_Testing	
Code-Based_Testing	Component_Testing
Compatibility_Testing	
Compliance_Testing	
Concurrency_Testing	
Conformance_Testing	
Clear-Box_Testing	
Development_Testing	
Efficiency_Testing	
Functional_Testing	
Functionality_Testing	
Glass-Box_Testing	
Interoperability_Testing	Integration_Testing
Installability_Testing	Integration Testing In The Large
	Integration Testing In The Small
	Interface_Testing
Logic-Coverage_Testing	
Logic-Driven_Testing	
	Module_Testing
	Non-Functional_Testing
	Operational Acceptance_Testing
	Operational Profile_Testing
	Operational_Testing
Performance_Testing	Program_Testing
Portability_Testing	
Procedure_Testing	
Production Acceptance_Testing	
Recoverability_Testing	
Recovery_Testing	
Regression_Testing	
Regulation_Testing	
Reliability_Testing	
Resource Utilization_Testing	
Robustness_Testing	
Safety_Testing	Static_Testing
Security_Testing	System Integration_Testing
Serviceability_Testing	System_Testing
Site_Acceptance_Testing	

Appendix A-1: STO Terms Classification

Specification-Based _Testing
Standards _Testing
Storage _Testing
Structurebased _Testing
Structural _Testing

User Acceptance _Testing

Unit _Testing

Volume _Testing

White-Box _Testing

Appendix A-1: STO Terms Classification

Activity		
Intrinsic		
Trace	Function	Attack
		Bebugging
Causal_Analysis Control_Flow_Analysis	Critical_Processes	Testing_
Data_Flow_Analysis Defect_Management Dynamic_Analysis	Daily_Build	Debugging Defect_Masking Desk_Checking Dynamic_Comparison
		Error_Seeding Fault_Attack Fault_Masking Fault_Seeding
	Maintenance	Resumption_Criteria
Static_Code_Analysis	Test_Design Test_Driven_Development Test_Execution Test_Execution_Automation Test_Implementation Test_Run	Software_Attack
Vertical_Traceability	Validation Verification	

Appendix A-1: STO Terms Classification

Activity					
Extrinsic					
Analysis	Control	Management	Report	Review	Procedure
				Ad-hoc_ Review Audit	Analyze
	Change_ Control Configuration_ Control	Change_ Management Configuration_ Identification Configuration_ Management		Configuration_ Auditing	
			Defect_ Report Deviation_ Report		
				Formal_ Review	
Hazard_ Analysis					Horizontal_ Traceability
Impact_ Analysis		Incident_ Management		Informal_ Review Inspection	Incident_ Logging Independence_ of_ Testing
				Management_ Review	

Appendix A-1: STO Terms Classification

		Problem_ Management	Problem_ Report	Peer_ Review	Post-execution_ Comparison Post-project_ Meeting
		Quality_ Assurance Quality_ Management			
Risk_ Analysis	Risk_ Control Risk_ Mitigation	Risk_ Identification Risk_ Management			Retrospective_ Meeting
Static_ Analysis				Structur ed_ Walkthr ough	
	Test_ Control Test_ Monitoring	Test_ Management Test_ Planning		Technic al_ Review	Test_ Closure Test_ Comparison Test_ Logging Test_ Phase Test_ Process Test_ Process_ Improvement (TPI) Test_ Recording Test_ Stage

Appendix A-1: STO Terms Classification

Version	
Control	
	Walkthrough

Appendix A-1: STO Terms Classification

Method.		
Technique	Approach	Practice
Action_ Word_ Driven_ Testing		Accessibility_ Testing
Algorithm_ Test		
Arc_ Testing		
Black-box_ Technique	Business_ Process-	Back-to-back_ Testing
Black-box_ Test_	based_ Testing	
Design_ Technique	Bottom-up_ Testing	
Boundary_ Value_ Analysis		
Boundary_ Value_ Testing		
Branch_ Condition_ Combination_ Testing		
Branch_ Testing		
Cause-effect_ Analysis	Complete_ Testing	Code_ Coverage
Cause-effect_ Graphing		Component_ Integration_ Testing
Checklist-based_ Testing		Condition_ Outcome
Classification_ Tree_ Method		Confidence_ Test
Condition_ Combination_ Testing		Configuration_ Testing
Condition_ Determination_ Testing		Confirmation_ Testing
Condition_ Testing		Conversion_ Testing
		Coverage_ Analysis
Data_ Driven_ Testing	Design-based_ Testing	Data_ Integrity_ Testing
Data_ Flow_ Testing		Database_ Integrity_ Testing
Decision_ Table_ Testing		Decision_ Condition_ Testing
Decision_ Testing		dd-path
Defect_ Based_ Technique		Dirty_ Testing
Defect_ Based_ Test_ Design_ Technique		Documentation_ Testing
		Dynamic_ Testing
Elementary_ Comparison_ Testing	Exhaustive_ Testing	Exception_ Handling
Equivalence_ Partitioning_ Error_ Guessing	Exploratory_ Testing	
Experienced-based_ Technique		
Experienced-based_ Test_ Design_ Technique		
Fault_ Tree_ Analysis	Failure_ Mode-and-	Field_ Testing

Appendix A-1: STO Terms Classification

(FTA) Finite_State_Testing Functional_Test_Design_Technique	Effect_Analysis (FMEA) Failure_Mode-Effect-and-Criticality_Analysis (FMECA) Functional_Integration Goal_Question_Metric	Function_Point_Analysis (FPA)
Heuristic_Evaluation		
		Incremental_Development_Model Incremental_Testing Instrumentation Intake_Test Integration Invalid_Testing Isolation_Testing Iterative_Development_Model
Keyword_Driven_Testing		
LCSAJ_Testing		Link_Testing Load_Testing
Modified_Condition_Design_Testing Modified_Multiple_Condition_Testing Multiple_Condition_Testing		Maintenance_Testing Maintainability_Testing Measurement Migration_Testing Monkey_Testing Mutation_Analysis Mutation_Testing
Non-functional_Test_Design_Techniques		Negative_Testing N-switch_Testing
Orthogonal_Array_Testing		
Pairwise_Testing Partition_Testing Path_Testing Process_Cycle_Test	Pair_Programming	path sensitizing pretest
Random_Testing Root_Cause_Analysis	Requirements-based_Testing Risk-based_Testing	Re-Testing
Scenario_Testing Specification-based_Technique Specification-based_Test_Design_Technique State_Transition	Session-based_Testing Software_Failure_Mode-and-Effect_Analysis (SFMEA) Software_Failure_Mode_Effect-and-Criticality_Analysis	Sanity_Test Session-based_Test_Management Scalability_Testing Scripted_Testing Smoke_Test Stress_Testing

Appendix A-1: STO Terms Classification

Testing	(SFMECA)	
Statement_ Testing	Software_ Fault_ Tree_	
Structure- based_	Analysis (SFTA)	
Technique	Software_ Usability_	
Structural_ Test_	Measurement_ Inventory	
Design_ Technique	(SUMI)	
Structure-based_ Test_		
Design_ Technique		
Statistical_ Testing		
Syntax_ Testing		
Systematic_ Test_ and_		
Evaluation_ Process		
Suitability_ Testing		
Test_ Case_ Design_	Top-down_ Testing	Test_ Point_ Analysis
Technique	Test_ Approach	(TPA)
Test_ Design_		
Technique		
Test_ Execution_		
Technique		
Test_ Specification_		
Technique		
Test_ Technique		
Use_ Case_ Testing		Usability_ Testing
User_ Scenario_ Testing		User_ Test
White-box_ Techniques		
White-box_ Test_		
Design_ Technique		
Wide_ Band_ Delphi		

Appendix A-2: STO Terms Properties

Object Property	Inverse Property
has Approach	isApproachOf
hasAutoProcess	isPerformedBy
hasCertification	Null
hasCheck	isCheckedBy
hasCode	isCodeOf
hasConfigure	isConfiguredBy
hasControl	isContoledBy
hasCriteria	isCriteriaOf
hasData	isDataOf
hasDebugger	Null
hasDocument	isDocumentOf
hasExtrinsicActivity	Null
hasFeature	Null
hasHardware	Null
hasImage	Null
hasIntrinsicActivity	Null
hasMeasurement	isMeasurementOf
hasModerate	isModerateBy
hasPlan	isPlannedBy
hasPractice	isPracticeBy
hasPurpose	isPurposeFor
hasRecord	Null
hasReference	isReferenceOf
hasReport	Null
hasResult	isResultsOf
hasReview	isReviewedBy
hasScope	isScopeOf
hasSignificance	Null
hasSoftware	Null
hasStandard	isStandardFor
hasTechnique	isTechniqueOf
hasTerminology	isTerminologyOf
hasTest	isTestedBy
hasText	isTextOf

Appendix A-3: STO Terms Data Properties

Data Property	Domain	Range
hasActualResults	Measurement	string
hasContributor	Test Case Suit	string
hasCreator	Artefact	string
hasCriteriaDescription	Criteria	string
hasExpectedResults	Text	string
hasGroupID	Test Case Suit	string
hadGuidTitle	Guide	string
hasImageID	Image	string
hasInputSpecification	Text	string
hasItemID	Null	string
hasNumberOfLines	Code	integer
hasPlanDescription	Plan	string
hasPostCondition	Test Case Suit	string
hasRatio	Features	Null
hasPrecondition	Test Case Suit	string
hasReleaseVersion	Null	string
hasReportDescription	Report	string
hasSoftwareID	Null	string
hasSource	Doc	Doc
hasStatus	Text	Boolean
hasTarget	Null	string
hasTermDescription	Term	string
hasTestCaseVersion	Null	string
hasTestDescription	Test Case Suite	string
hasTestObjective	Test Case Suite	string
hasTestScript	Null	string
hasTestTitle	Text	string
hasTestType	Task Testing	string
hasTestID	Individual	string
hasTitle	Null	string

Appendix A-3: STO Terms Data Properties

isInfectedCode	Code	Boolean
----------------	------	---------

Appendix B: STCMS Documentation

1. Introduction

This Software Requirements Specification (SRS) is written to identify the requirement of **Semantic Web-based Test Case Management System**. This project is implemented to verify the fulfilment of PhD research on Semantic Web & Test Case Management System supported by Semantic Technology.

1.1.Purpose

The SRS is to clearly identify the requirements that need to be included in the system. The SRS is used in further development stages. It is very vital to state every requirement precisely. Each requirement introduces the most important issue of the system functionality. The findings of the SRS are the system main functionalities.

1.2.Scope

The system uses the inspiration of semantic web based system to represent and supercharge the testing case management. The system is engine with the support of semantic technology. There are 2 main pivots our system discerned its requirement out of. The first pivot is the recommended standards by W3C for the semantic web layered; this was by referring to the main sources stored in their website. The second pivot is the test case management system requirements from users (testers) point of view; this was by studying an existing software test management system.

1.3. Out of Scope

The proposed software will not cover the registration, expiry dates, and activating the registered *clients*.

1.4. Application of the Software

The application will enable wide range of *industry* and *individuals* to interact with test cases. The usage of the application will facilitate the clients test process management. It will manage the execution of test cases in the system. The list of the goals that can be achieved includes: task managements, powerful searches, increase business opportunities.

1.4.1. Task Managements

The client will be able to use the application to perform specific tasks; for example search on test cases from all over the world. These tasks can be performed at the client suitable time. Creating, Monitoring, and other tasks can be delegated to specific people to do. The overall administration grants will enable the client's user to carry out their task in independent way. The main benefit gained from such task is that the ability to modify and add new fields as they occur in the future. Another benefit is ease of update of these fields to match any further field renaming.

1.4.2. Powerful searches

The search is used by any user. The user can search in semantic way to get the best hits for the test cases. The search will help the tester and user to find which test case to be reused, and which can suit the current test requirement.

1.4.2.1. Increase Business Opportunities

Using this application client will have an opportunity to increase its business by managing and researching a large number of test cases.

1.5. Definitions, acronyms, and abbreviations

This document uses the following terms and abbreviation

Abbreviation	Description
SRS	Software Requirement Specification
Client	The business which required the system to be developed
User	Any type of users who uses the systems
UML	Unified Modeling Language
STO	Software Testing Ontology

1.6. References

Guide:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=278253&isnumber=6883>

Practice:<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574&isnumber=15571>

2. Overall Descriptions

2.1.Product perspective

This System is an independent System. The system is divided into three major modules. These modules will cooperate with each other to perform the required tasks.

2.2.Product functions

The main functions of the product will be as follows:-

- ❖ Creating Test Cases
- ❖ Managing Test Cases
- ❖ Reporting Test Cases in IEEE standard
- ❖ Save the Test Cases Semantically
- ❖ Search Test Cases Semantically

2.3.User characteristics

The general characteristics of the intended users of the System should be:-

- ❖ Test Planner
- ❖ Test Engineer
- ❖ QA Analyst
- ❖ Test Manager

3. Specific Requirements

3.1. External interface requirements

- ❖ User interface
- ❖ Software interface
- ❖ Communication interface

3.2. Functional requirements

The Functional requirements describe the functions of the systems in a form of use cases as shown in the following UML diagram and use case description.

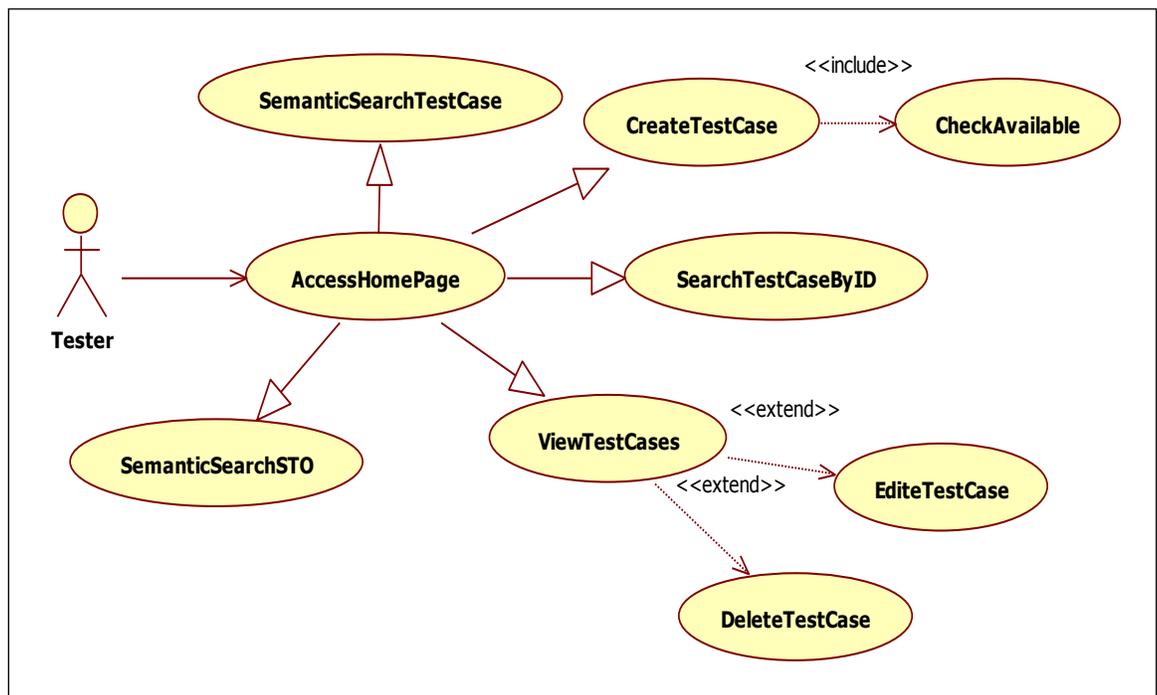


Figure 3.2-1 SWTCMS Use Cases Diagram

3.2.1. Access Homepage

i. Description

This use case is used by user to **access** the main page of the system.

ii. Flow of Event(s)

1. The user launch the system browser
2. The system displays the homepage
3. The system provides user to do other functions on the system
4. The use case continues

3.2.2. Create Test Case

i. Description

This use case is used by user to **create** test cases in the system.

ii. Flow of Event(s)

1. The user enters the test case specification identifier
2. The user enter Test Case details (**System Check Availability**)
3. The user specify the pre requirement for test case execution
4. The user enter these details (**Input & Expected Output**)
5. The system prompts user to create the test case or rest the form
6. System records the Execution History [**Date, Version**] & save
7. The use case ends

3.2.3. Search Test Case by ID

i. Description

This use case is used by user to **search** test cases if **ID** is known.

ii. Flow of Event(s)

1. The user enter test case ID
2. The system search for matching ID
3. The system view results
4. The use case ends

3.2.4. Semantic Search Test Case

i. Description

This use case is used by user to **search** test cases **semantically**.

ii. Flow of Event(s)

1. The user enter key word to search
2. The system navigate the key word with the STO
3. The system view available matching in the STO to help user find more related key words
4. Upon user word selection system search test cases & display results
5. The use case ends

3.2.5. View Test Case

i. Description

This use case is used by user to **view** available test cases.

ii. Flow of Event(s)

1. The system prompts user to view or delete available test cases
2. If user **delete** test case system proceeds upon confirmation
3. If user select details of test case
4. The system displays test case details and provide **edit** facility
5. The user **edit** test case system displays data in Create Test Case Form [3.5.1: Create Test Case]
6. The use case ends

3.2.6. View Software Test Ontology

i. Description

This use case is used by user to **browse** software testing **ontology**

ii. Flow of Event(s)

1. The system displays software testing ontology
2. The system provide user to browse by one of the following
 - a. Class
 - b. Properties
 - c. Individual
3. The system displays details of the selected option
4. The use case continues

3.2.7. STO Concept Search

i. Description

This use case is used by user to **search** for **STO concepts**.

ii. Flow of Event(s)

1. The user enter key word to search
2. The system navigate the key word with the STO concepts
3. The system view available matching in the STO to help user find more related key words
4. Upon user word selection system search & display results
5. The use case ends

3.2.8. STO Properties Search

i. Description

This use case is used by user to **search** for **STO properties**.

ii. Flow of Event(s)

1. The user enter key word to search
2. The system navigate the key word with the STO properties
3. The system view available matching in the STO to help user find more related key words
4. Upon user word selection system search & display results
5. The use case ends

3.2.9. STO Individual Search

i. Description

This use case is used by user to **search** for STO **individuals**.

ii. Flow of Event(s)

1. The user enter key word to search
2. The system navigate the key word with the STO individuals
3. The system view available matching in the STO to help user find more related key words
4. Upon user word selection system search & display results
5. The use case ends

3.2.10. Query Software Test Ontology

i. Description

This use case is used by user to **query** the software testing **ontology**

ii. Flow of Event(s)

1. The user enter query syntax
2. The system inquiry the ontology
3. The system display results
4. The use case ends

3.3.Performance requirements

The system will be a semantic web base solution. It will interact with the users.

The system will be able to handle requests simultaneously. The speeds of performing each request will depend on two items are:

- ❖ Internet speed: there will be no control over on the network.
- ❖ Servers speed: most requests will be handled within less than 30 seconds.

The performance can be enhanced if the client rented a leased line with minimum of 128 bit. As far as more request start flying to the server the client is requested to upgrade the line speed.

3.4.Logical Database requirements

The proposed system is capable to store information about the test cases in a database that defines relationships between different test cases terms.

3.5.Design constraints

The solution will be used in a web based environment. It will be better if the design is oriented to an Object-Oriented Design. In case of using an Internet Service Provider Hosting (ISPH) to host the site, then the hardware is out of control. Firewall configuration might be another issue to be looked after.

3.6. Software system attributes

- **Availability:**

The system will be available to the every one who can reach the internet. In addition the time availability depends on the ISPH and the Internet. It is possible to have an additional backup system. This option depends on the ISPH used software facilities.

- **Security:**

The system will maintain the security to the level of the application. The actual data is laying in the ISPH servers. This way of hosting will enable access the data at application level. The database might be accessed by authorized administrators of the ISPH. In addition the Open Source Database engine may not supports high level security.

- **Portability:**

The solution is portable to different platforms. It can be use in Windows, Mac, UNIX, or Linux environment. The web clients can still see the same layout and the same results. The main reason is that the communication might be a standard recommended by W3C.

- **Usability:**

The solution is using friendly Graphical User Interfaces (GUI) which does not require any knowledge or guide to be used.

4. Traceability Matrix

#	Name of ID	Requirements
1.	SWTCMS_SRS_100.01	Access Homepage
2.	SWTCMS_SRS_100.02	Create Test Case
3.	SWTCMS_SRS_100.03	Search Test Case by ID
4.	SWTCMS_SRS_100.04	Semantic Search Test Case
5.	SWTCMS_SRS_100.05	View Test Case
6.	SWTCMS_SRS_101.01	View Software Test Ontology
7.	SWTCMS_SRS_101.02	STO Concept Search
8.	SWTCMS_SRS_101.03	STO Properties Search
9.	SWTCMS_SRS_101.04	STO Individual Search
10.	SWTCMS_SRS_101.05	Query Software Test Ontology

1. INTRODUCTION

This document defines the activities and responsibilities of research on Semantic Web & Test Case Management System with regard to the study, design, development, qualification, testing and delivery of the software concerning the SWTCMS System Application.

1.1.PURPOSE

The application to be developed shall enable searching test cases semantically and enhances the scope of the information sharing. For SWTCMS, the application will be web application with automated workflow for initiating test case processing, which will further improve the efficiency and services of software testing.

1.2.SCOPE

The application provides features to capture the information create, update and delete the transaction in order to provide full management for new test cases available in the System.

1.3.DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

This document uses the following terms and abbreviation.

Abbreviation	Description
SDD	System Design Description
SWTCMS	Semantic web testing case management system
MVC	Model-View-Controller

1.4. REFERENCES

Guide

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=741934&isnumber=16019>

2. ARCHITECTURAL DESIGN

SWTCMS System will be developed by applying Model-View-Controller (MVC) architecture. The MVC architecture is a widely-used architectural approach for interactive applications. It divides functionality among objects involved in maintaining and presenting data to minimize the degree of coupling between the objects. The architecture maps traditional application tasks (input, processing, and output) to the graphical user interaction model. They also map into the domain of multitier Web-based enterprise applications.

The MVC architecture divides applications into three layers (model, view, and controller) and decouples their respective responsibilities. Each layer handles specific tasks and has specific responsibilities to the other areas.

- A **model** represents business data and business logic or operations that govern access and modification of this business data. Often the model serves as a software approximation to real-world functionality. The model notifies views when it changes and provides the ability for the view to query the model about its state. It also provides the ability for the controller to access application functionality encapsulated by the model. In SWTCMS, model will represent the relational database and the semantic test case data.

-
- A **view** renders the contents of a model. It accesses data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller.

 - A **controller** defines application behaviour. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a stand-alone GUI client, user inputs include button clicks and menu selections. In a Web application, they are HTTP GET and POST requests to the Web tier. A controller selects the next view to display based on the user interactions and the outcome of the model operations. An application typically has one controller for each set of related functionality. Some applications use a separate controller for each client type, because view interaction and selection often vary between client types.

The relationship described is shown in the following figure.

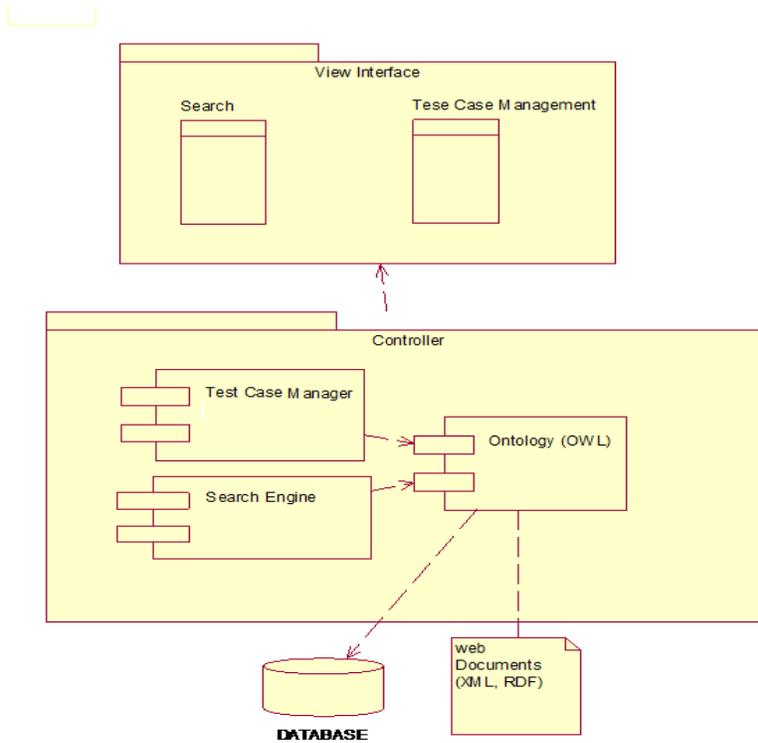
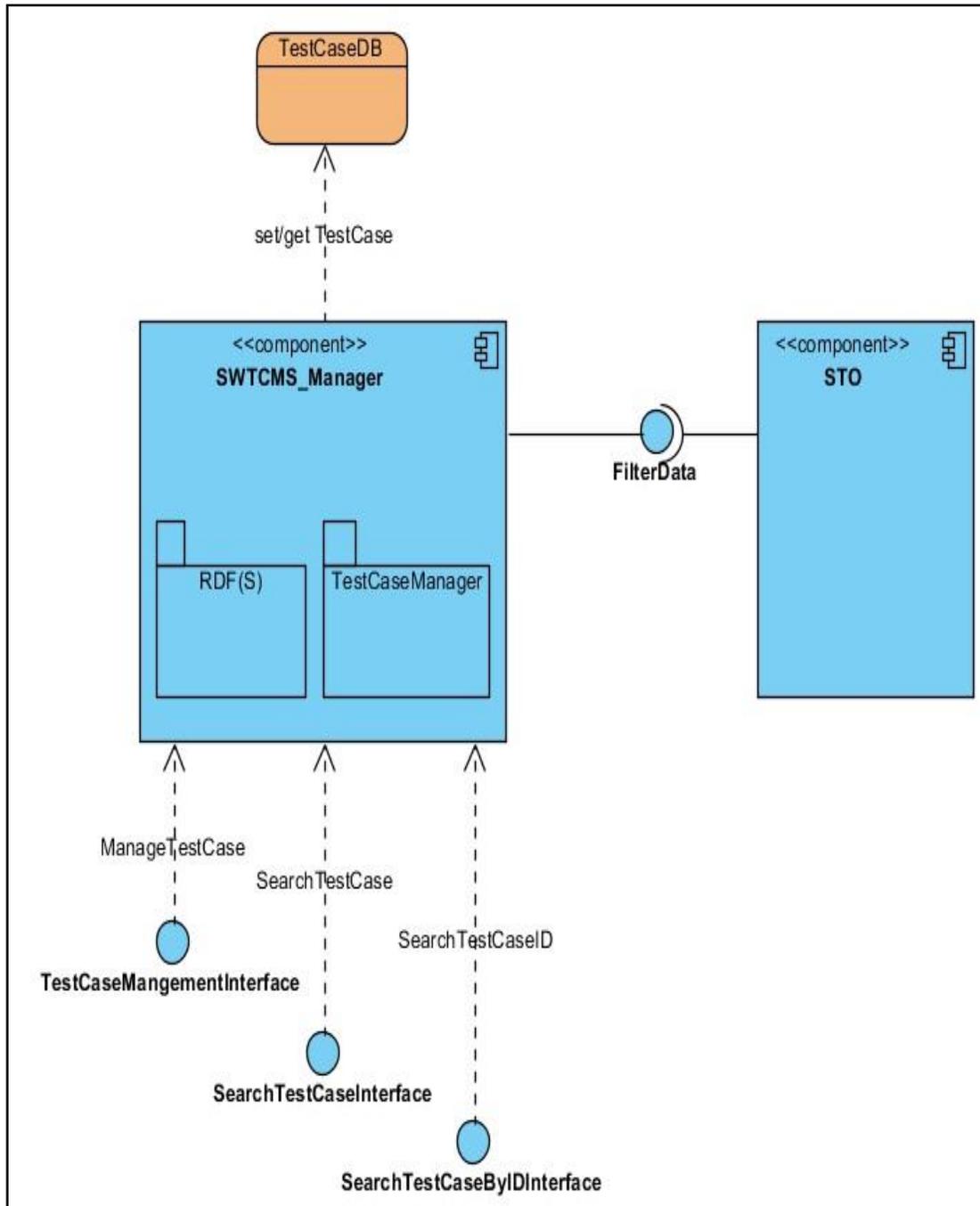


Figure 2-1 SWTCMS Architecture Diagram

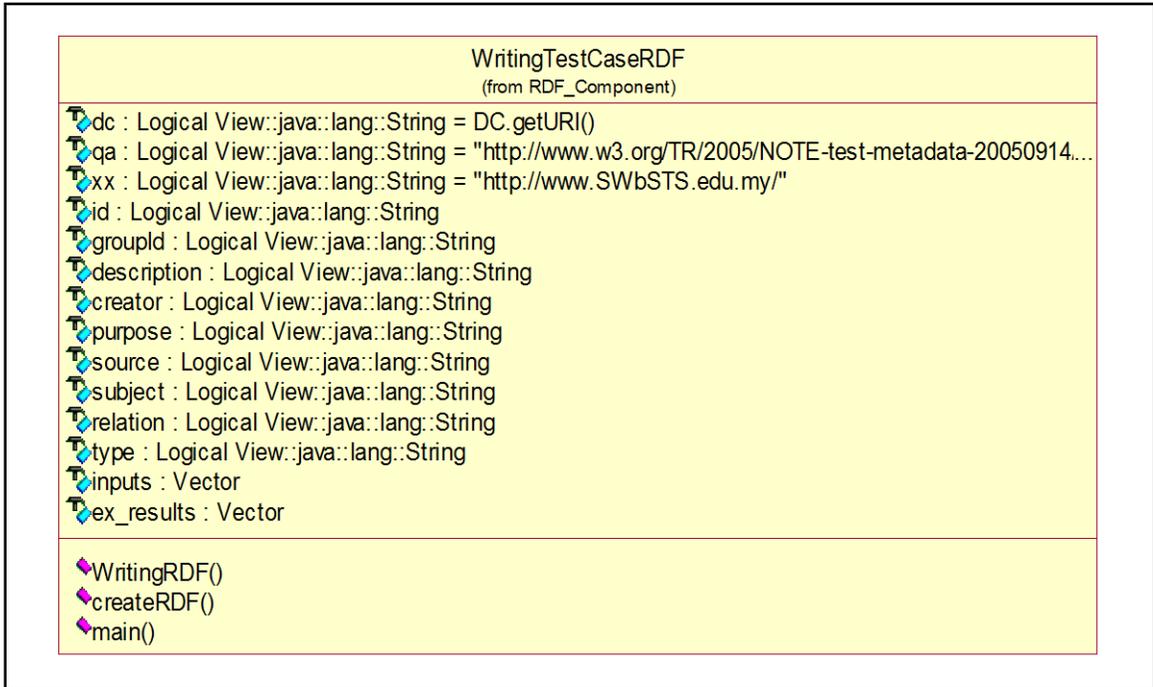
3. DETAILED DESIGN

The internal organizational structure and detail description in the SWTCMS Application as describe below.

3.1. Component DESIGN

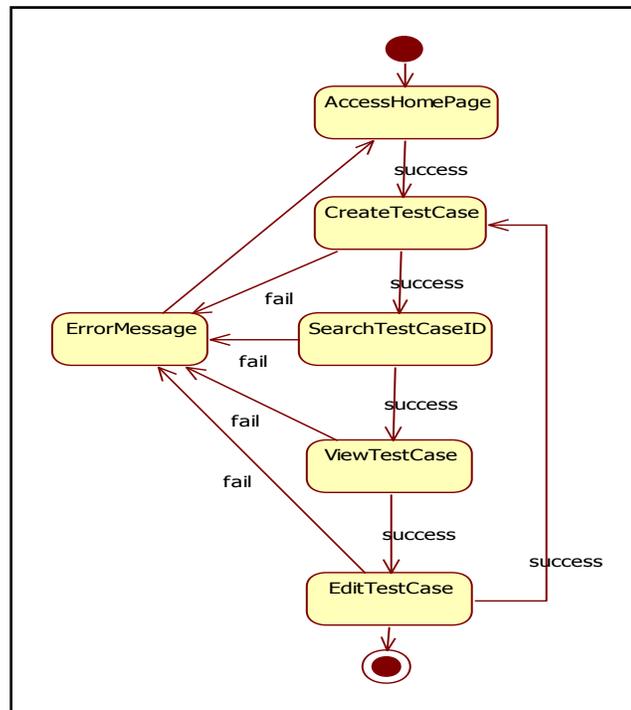


3.2.CREATE TEST CASE CLASS DIAGRAM

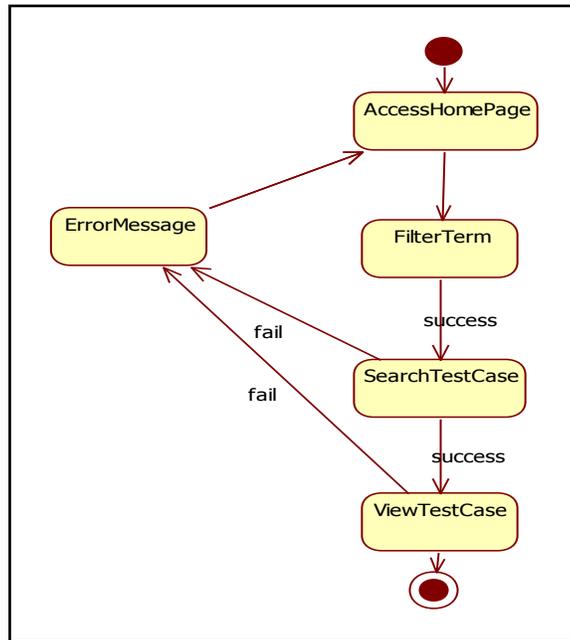


3.3. STATE DIAGRAM

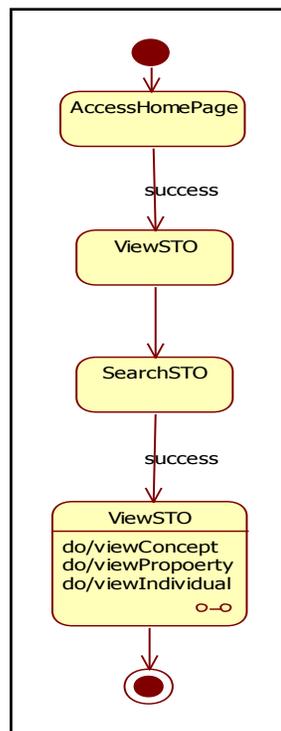
3.3.1. Test Case Management



3.3.2. Test Cases Semantic Search



3.3.3. SOFTWARE TEST ONTOLOGY



4. TRACEABILITY MATRIX

#	Name of ID	Covers in SRS	Description
---	------------	---------------	-------------

		SWTCMS_SRS_100.01	
11.	SWTCMS_SDD_100	SWTCMS_SRS_100.02	Test Case Management State Diagram
		SWTCMS_SRS_100.03	
		SWTCMS_SRS_100.05	
12.	SWTCMS_SDD_101	SWTCMS_SRS_100.04	Test Cases Semantic Search State Diagram
13.	SWTCMS_SDD_102	SWTCMS_SRS_101.01	Software Ontology Diagram
		SWTCMS_SRS_101.02	
		SWTCMS_SRS_101.03	
		SWTCMS_SRS_101.04	
		SWTCMS_SRS_101.05	

System: SWTCMS

Test Case No.	1. Access Homepage	Test Case Version	Version 1.0
----------------------	---------------------------	--------------------------	-------------

Test Title	SWTCMS_AccessHomePage
Test Objective	To access the home page of SWTCMS
Pre-requisite	Server should be on
Tester	Mansoor

Step	Description	Expected Result	Remarks
4.	Launch System website on browser	Home screen will be displayed	
5.	Move mouse on home screen bar	Title colour changes to get ready to be accessed	

System: SWTCMS

Test Case No.

2. Create Test Case

Test Case Version

Version 1.0

Test Title

SWTCMS_CreateTestCase

Test Objective

To create a test case and save it in the system

Pre-requisite

Tester

Mansoor

Step	Description	Expected Result	Remarks
1.	Click on Create Test Case from the main home page/	Create Test Case Form will be displayed	
2.	Enter the Test Case ID with no space	System checks the ID and notify actor if space was provided	
3.	Click on Add Input	System provides input and expected results text field.	
4.	Click on Submit button	System save test case data and notify actor with confirmation message.	

System: SWTCMS

Test Case No.

3. Rest Test Case Form

Test Case Version

Version 1.0

Test Title

SWTCMS_RestTestCaseForm

Test Objective

To rest the text fields in the Create Test Case Form

Pre-requisite

Data had been filled in

Tester

Mansoor

Step	Description	Expected Result	Remarks
1.	Click on Create Test Case from the main home page/	Create Test Case Form will be displayed	
2.	Key in data in the text field	Data in the text field	
3.	Click reset button	Text filed will empty the text	

System: SWTCMS

Test Case No.

4. Search Test Case ID

Test Case Version

Version 1.0

Test Title

SWTCMS_SearchTestCaseID

Test Objective

To search a specific Test Case if ID is known to the actor

Pre-requisite

Data filled in

Tester

Mansoor

St ep	Description	Expected Result	Remarks
1.	Click on Search Test Case by ID	Search Test Case ID form will be displayed	
2.	Key in Test Case ID and click Search button	Search results will be displayed	
3.	Click on Test Case Details	Test case details will be displayed	

System: SWTCMS

Test Case No.

5. Semantic Search Test Case

Test Case Version

Version 1.0

Test Title

SWTCMS_SearchTestCase

Test Objective

To search semantically Test Case according search Term
--

Pre-requisite

Test Cases availability in the system database
--

Tester

Mansoor

Step	Description	Expected Result	Remarks
1.	Click on Search Test Case	Search Test Case Form will be displayed	
2.	Key in search term	Semantic drop down list show available terms match with search term	
3.	Click search button	Search results will be displayed	
4.	Click on Test Case Details	Test case details will be displayed	

System: SWTCMS

Test Case No.	6. View Software Testing Term	Test Case Version	Version 1.0
----------------------	--------------------------------------	--------------------------	-------------

Test Title	SWTCMS_ViewSoftwareTestingTerm
Test Objective	To view Software Testing Terms to help search
Pre-requisite	Test Cases availability in the system database
Tester	Mansoor

Step	Description	Expected Result	Remarks
1.	Click on View Test Case	View Test Case List will be displayed	
2.	Click on Navigation Bar	Terms related to Software Testing will be displayed	
3.	Select required term	Term selected will be displayed in search field	
4.	Click search button	Search results will be displayed	

System: SWTCMS

Test Case No.

7. View Test Case List

Test Case Version

Version 1.0

Test Title

SWTCMS_ViewTestCaseList

Test Objective

To view Test Case available in the System

Pre-requisite

Test Cases availability in the system database

Tester

Mansoor

Step	Description	Expected Result	Remarks
1.	Click on View Test Case	View Test Case List will be displayed	
2.	Click on Test Case Details	Test case details will be displayed	
3.	Click on Test Case Delete	Reconfirmation Message will be displayed	
4.	Click Yes/ No	Test Case will/ will not be deleted	

System: SWTCMS

Test Case No.	8. STO Concept Search	Test Case Version	Version 1.0
----------------------	------------------------------	--------------------------	-------------

Test Title	SWTCMS_SOTConceptSearch
Test Objective	To search the concept and classes of Software Testing Ontology
Pre-requisite	
Tester	Mansoor

Step	Description	Expected Result	Remarks
1.	Click on Software Testing Ontology	Software Testing Ontology Form will be displayed	
2.	Click on Classes Tap	Class Form displays All classes and concepts available for Software Testing	
3.	Click on Tree View or Navigation Bar	Tree view or Navigation view for Concepts will be displayed	
4.	Click on Concept	Description of the Concept with its relations will be displayed	

System: SWTCMS

Test Case No.

9. STO Properties Search

Test Case Version

Version 1.0

Test Title

SWTCMS_SOTPropertiesSearch

Test Objective

To search the properties of Software Testing Ontology

Pre-requisite

Tester

Mansoor

Step	Description	Expected Result	Remarks
1.	Click on Software Testing Ontology	Software Testing Ontology Form will be displayed	
2.	Click on Properties Tap	Properties Form displays All Properties available for Software Testing	
3.	Click on Tree View or Navigation Bar	Tree view or Navigation view for Properties will be displayed	
4.	Click on Properties	Description of the Properties with its relations will be displayed	

System: SWTCMS

Test Case No.	10. STO Individual Search	Test Case Version	Version 1.0
Test Title	SWTCMS_SOTIndividualSearch		
Test Objective	To search the Individual of Software Testing Ontology		
Pre-requisite			
Tester	Mansoor		

Step	Description	Expected Result	Remarks
1.	Click on Software Testing Ontology	Software Testing Ontology Form will be displayed	
2.	Click on Individual Tap	Individual Form displays All Individual available for Software Testing	
3.	Click on Tree View or Navigation Bar	Tree view or Navigation view for Individual will be displayed	
4.	Click on Individual	Description of the Individual with its relations will be displayed	

Appendix C: Test Cases Data

System: FSKTM PERSONALIZED WEBSITE

Test Case No.	11. Access Homepage	Test Case Version	Version 1.0
----------------------	----------------------------	--------------------------	----------------

Test Title	FPW_AccessHomePage
Test Objective	To access the home page of FSKTM PERSONALIZED WEBSITE
Pre-requisite	Server should be on
Tester	Faduma

Step	Description	Expected Result	Remarks
6.	Launch System website on browser	Home screen will be displayed	
7.	Move mouse on home screen bar	Links are ready to be accessed	

Test Case No.	12. Login	Test Case Version	Version 1.0
----------------------	------------------	--------------------------	----------------

Test Title	FPW_Login
Test Objective	To login with a pre registered user ID
Pre-requisite	Lunch FSKTM website
Tester	Faduma

Step	Description	Expected Result	Remarks
5.	Click on login the main home page	Login form will display	
6.	Key in the correct user ID and password	System will display the main home page under the user's ID	

Test Case No.	13. Personalize Background	Test Case Version	Version 1.0
----------------------	-----------------------------------	--------------------------	-------------

Test Title	FPW_Personalize_Background
Test Objective	To personalize backgrounds and save it with the user ID
Pre-requisite	Login with registered user ID
Tester	Faduma

Step	Description	Expected Result	Remarks
1.	Click on Background from the main home page/	Background list will be displayed	
2.	Select the preferred Background	System will display the background and save it under the user's ID	

Test Case No.	14. Personalize Layout	Test Case Version	Version 1.0
Test Title	FPW_Personalize_Layout		
Test Objective	To personalize layout and save it with the user ID		
Pre-requisite	Login with registered user ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
4.	Click on Background from the main home page/	Background list will be displayed	
5.	Select the preferred Background	System will display the background and save it under the user's ID	

Test Case No.	15. Rearrange Panels	Test Case Version	Version 1.0
Test Title	FPW_Rearrange_Panels		
Test Objective	To rearrange panels and save it with the user ID		
Pre-requisite	Login with registered user ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
5.	Rearrange panels by drag and drop on the main home page/	Personalized panels will be displayed and saved under the user's ID	
6.	Click on Home page	Panels in personalized order will be displayed	

Test Case No.	16. Personalize Panels	Test Case Version	Version 1.0
Test Title	FPW_Personalized_Panels		
Test Objective	To personalize panels and save them under the user ID		
Pre-requisite	Login with Admin ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
5.	Hide, show less links and show more links in each panel on the main home page	Actions will be displayed and saved under the user's ID	
6.	Click on Home page	Panels in personalized order will be displayed	

Test Case No.	17. Quick Links	Test Case Version	Version 1.0
Test Title	FPW_Quick_Links		
Test Objective	To create quick links and save them under the user's ID		
Pre-requisite	Login with Admin ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
1.	Select links from different panels and click save	links will be displayed under quick links panel and saved under the user's ID	
2.	Click on Home page	Updated Quick links Panels is displayed	

Test Case No.	18. System Management	Test Case Version	Version 1.0
Test Title	FPW_System_Managment		
Test Objective	To check admin functions		
Pre-requisite	Login with Admin ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
1.	Click on user ID	Registered users list will be displayed	
2.	Click on add	Menu to add users will display	
3.	Add users credentials	Added credentials will be saved in the database	
4.	Edit users credentials	Registered users will be edited.	
5.	Delete users credentials	Registered users will be deleted.	

Test Case No.	19. Logout	Test Case Version	Version 1.0
Test Title	FPW_Logout		
Test Objective	To logout from FSKTM website		
Pre-requisite	Login with registered user ID		
Tester	Faduma		

Step	Description	Expected Result	Remarks
5.	Click on logout	Home screen will be displayed	
6.	Move mouse on home screen bar	Links are ready to be accessed	

System: iLogger

Test Case No.	1. iLogger_iSmart_100.01	Test Case Version	Version 1.0
----------------------	---------------------------------	--------------------------	----------------

Test Title	iRegisterService
Test Objective	It will allow admin to register the particular module.
Pre-requisite	Server should be on
Tester	Developers

Step	Description	Expected Result	Remarks
1.	Launch the system.	Main interface displayed.	
2.	Enter URL	Particular module is registered or not.	

Test Case No.	2. iLogger_iSmart_100.02	Test Case Version	Version 1.0
----------------------	---------------------------------	--------------------------	----------------

Test Title	iUnregisterService
Test Objective	It will allow admin to unregister the particular module
Pre-requisite	Server should be on
Tester	Developers

Step	Description	Expected Result	Remarks
1.	Launch the system	Main interface displayed	
2.	Enter URL	Particular module is unregistered or not unregistered.	

Test Case No.	3. iLogger_iSmart_100.03	Test Case Version	Version 1.0
----------------------	---------------------------------	--------------------------	----------------

Test Title	iIdentifyFreeService
Test Objective	It will identify free service/module and assign task
Pre-requisite	Module should be registered
Tester	Developers

Step	Description	Expected Result	Remarks
1.	Enter the module type	Module type match with the suitable process	
2.	Check the availability of the process matched	Particular Process is Free to Process, NULL if all busy	

Test Case No.	4. iLogger_iSmart_100.04	Test Case Version	Version 1.0
----------------------	---------------------------------	--------------------------	----------------

Test Title	iSynchronizeVersion
Test Objective	It will synchronize the modules version number.
Pre-requisite	Modules should be registered
Tester	Developers

Step	Description	Expected Result	Remarks
1.	True or false	System will synchronize the version	

Test Case No.	5. iLogger_iModule_101.01	Test Case Version	Version 1.0
Test Title	iCheckBusy		
Test Objective	It will check whether the particular module is busy or free and return the status		
Pre-requisite	A component will produce an URL		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	System will get the URL from one of the component	The URL is checked whether it is busy or free.	

Test Case No.	6. iLogger_iModule_101.02	Test Case Version	Version 1.0
Test Title	iCheckLive		
Test Objective	It will check whether the particular module is live or down and return the status.		
Pre-requisite	A component/module will produce an URL		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	System will get the URL from one of the component/module	The URL is checked whether it is live or down.	

Test Case No.	7. iLogger_iModule_101.03	Test Case Version	Version 1.0
Test Title	iCheckVersion		
Test Objective	It will check the particular Version Number and return the Version Number in String		
Pre-requisite	A component/module will produce an URL		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	System will get the URL from one of the component/module	The URL is checked what the version is using.	

Test Case No.	8. iLogger_iModule_101.04	Test Case Version	Version 1.0
Test Title	iRecoverErro		
Test Objective	It will check whether the process is completed or failed and return the status		
Pre-requisite	Server should be started		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	Recover errors.	Transforms back to Live state.	

Test Case No.	9. iLogger_iModule_101.05	Test Case Version	Version 1.0
Test Title	iRecordProcessTime		
Test Objective	It will record the transaction of process time and return the time.		
Pre-requisite	A component/module will produce a Path		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	System will get the Path from one of the component/module	The Path is checked whether the whole process is completed or failed.	

Test Case No.	10. iLogger_iFile_102.01	Test Case Version	Version 1.0
Test Title	iAcceptFile		
Test Objective	Must accept file (preferably zipped file) from clients' side machine.		
Pre-requisite	Server should be on and user must be logged in		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	Waiting for zipped file from clients	Server ready to accept the file.	

Test Case No.	11. iLogger_iFile_102.02	Test Case Version	Version 1.0
Test Title	iCount		
Test Objective	Must return process count (% of process/upload byte/sec/total time use to upload)		
Pre-requisite	Server should be on and user must be logged in		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	System accept preferably zipped	Particular Process Count	

Test Case No.	12. iLogger_iFile_102.03	Test Case Version	Version 1.0
Test Title	iUnzipFile		
Test Objective	Must perform action of unzipping file from the clients file		
Pre-requisite	Server has accepted the file		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	Reads the file.	Display the files found in archive.	

Test Case No.	13. iLogger_iFile_102.04	Test Case Version	Version 1.0
Test Title	iRejectFile		
Test Objective	Must able to reject unwanted file which found in the unzipped file.		
Pre-requisite	Server has accepted the file		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	Check the file type.	Display all the file type.	
2.	Select file to be moved	Reject file except .log & .txt file	

Test Case No.	14. iLogger_iFile_102.05	Test Case Version	Version 1.0
Test Title	iMoveFile		
Test Objective	Must transfer the file to specific folder that is Reject File folder.		
Pre-requisite	Reject file moves to reject folder		
Tester	Developers		

Step	Description	Expected Result	Remarks
1.	Save to database.	.log and .txt file save to database.	

Test Case No.	15. iLogger_iFile_102.06	Test Case Version	Version 1.0
Test Title	iZipFile		
Test Objective	Must compress/zip required files		
Pre-requisite	Server has accepted the file		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	System receives log file	Zipped the log file	

Test Case No.	16. iLogger_iPattern_103.01	Test Case Version	Version 1.0
----------------------	------------------------------------	--------------------------	----------------

Test Title	iFilterPattern
Test Objective	Collect the log files from different machines analyse and filter the same pattern log entries into a pattern text file.
Pre-requisite	Server has accepted the log file
Tester	Developers

S t e p	Description	Expected Result	Remarks
1.	Collect the log file from different machines.	Server ready to filter the log files collected.	
2.	Analyze and filter the same pattern log entries.	Output into pattern text files.	

Test Case No.	17. iLogger_iPattern_103.02	Test Case Version	Version 1.0
----------------------	------------------------------------	--------------------------	----------------

Test Title	iNormalizePattern
Test Objective	Standardize multiple log entries of the same pattern to a preferred pattern
Pre-requisite	Server has accepted the log file
Tester	Developers

S t e p	Description	Expected Result	Remarks
1.	Identify similar log entries	Server ready to combine similar log entries.	
2.	Combine multiple similar log entries into one entry	Output the log entries into a pattern text file.	

Test Case No.	18. iLogger_iPattern_103.03	Test Case Version	Version 1.0
----------------------	------------------------------------	--------------------------	----------------

Test Title	iIdentifyPattern
Test Objective	Search the log files for the log entries containing keyword specified by user and output them into a pattern text file once confirmed by user.
Pre-requisite	Server has accepted the log file
Tester	Developers

S t e p	Description	Expected Result	Remarks
------------------	-------------	-----------------	---------

S t e p	Description	Expected Result	Remarks
1.	Prompt user for keyword.	Display log entries containing the specified keyword.	
2.	User confirms to set the keyword as default log pattern.	System ready to filter files.	
3.	System filters the specified pattern.	Output the log entries into a pattern text file.	

Test Case No.	19. iLogger_iStatistic_104.01	Test Case Version	Version 1.0
Test Title	iCalculateFrequency		
Test Objective	Calculate the frequency of a particular process count within certain period		
Pre-requisite	Receive pattern.txt file from iPattern		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	Open the pattern .txt file.	Display the pattern.txt file.	
2.	Calculate the frequency.	Successfully count the frequency of pattern.	

Test Case No.	20. iLogger_iStatistic_104.02	Test Case Version	Version 1.0
Test Title	iCalculateProcessTime		
Test Objective	Calculate the process time in between a process ends and the start of a new process		
Pre-requisite	Receive pattern.txt file from iPattern		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	Open the pattern text file	Display the pattern .txt file.	
2.	Check for sections of process occurrence.	No reaction	
3.	Calculate the process time of each sections	Successfully count the process time and display the time.	

Test Case No.	21. iLogger_iStatistic_104.03	Test Case Version	Version 1.0
Test Title	iCalculateStandardDeviation		
Test Objective	Calculate the standard deviation transaction recorded from each pattern.		
Pre-requisite	Receive pattern.txt file from iPattern		
Tester	Developers		

S t e p	Description	Expected Result	Remarks
1.	Receive user prompt for the axis (time interval).		
2.	Use the min frequency for the x axis.		
3.	The information in x and y axis are used to plot the diagram of standard deviation	Show the diagram of standard deviation based on its statistics.	

Test Case No.	22. iLogger_iStatistic_104.04	Test Case Version	Version 1.0
Test Title	iCalculateHistogram		
Test Objective	Calculate the histogram transaction recorded from each pattern.		

Pre-requisite
Tester

Receive pattern.txt file from iPattern
Developers

S t e p	Description	Expected Result	Remarks
1.	Receive user prompt for the y axis (time interval).		
2.	Use the frequency for the x axis		
3.	The information in x and y axis are used to plot the diagram of standard deviation	Show the diagram of histogram based on its statistics	

System: MFIT FoodReg Chicken Boiler

Test Script No.	1. MFIT_Login	Test Script Version	1.0
------------------------	----------------------	----------------------------	-----

Test Title	Login (Timer name : "Login")
Test Objective	Login to the system.
Pre-requisite	Valid username and password.

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the "Username" field.	Username is entered.	Compulsory e.g. <mdec>
4.	Enter password in the "Password" field	Password is entered.	Compulsory e.g. <mdec01>
5.	< Start Block - Login > Click on Login button.	Login is successful and "Personal Information" page is displayed. < Stop Block - Login >	
6.	Click "Logout" button on the top right hand corner of the MFIT FoodReg main page.	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	2. MFIT_AddWorker	Test Script Version	1.0
------------------------	--------------------------	----------------------------	-----

Test Title	AddWorker (Timer name : "add_wkr")
Test Objective	Add new workers information into the system
Pre-requisite	Valid username and password

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/mdtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the "Username" field.	Username is entered.	Compulsory e.g.<mdec>
4.	Enter password in the "Password" field	Password is entered.	Compulsory e.g.<mdec01>
5.	Click on "General Data" tab	"General Data" page is displayed.	
6.	Click on "Your Company" link	"Your Company" page is displayed.	
7.	Click on "Add A New Worker" link	"Create Worker" page is displayed.	
8.	Enter Known as in the "Known as" field	Known as is entered.	Compulsory e.g.<sadc>
9.	Enter First name in the "First name" field	First name is entered.	Recommended e.g.<sadc>
10.	Enter Surname in the "Surname" field	Surname is entered.	Recommended e.g.<sadc>
11.	Select Company in "Company" field	Company is selected	Recommended e.g.<DBE Food Processing>
12.	Select User in the "Insert additional person profile" field	User is selected.	Compulsory e.g.<User>
13.	Enter Username in the "Username" field	Username is entered.	Compulsory Unique e.g.<sadc>
14.	Enter New Password in the "NEW PASSWORD" field	New Password is entered.	Compulsory Unique (min 6 char with numeric) e.g.<password1>
15.	Enter Retype New Password in the "RETYPE NEW PASSWORD" field	Retype New Password is entered.	Compulsory Unique (min 6 char with numeric) e.g.<password1>
16.	Select Status in "Status" filed	Status is selected.	Compulsory e.g.<active>

S t e p	Description	Expected Result	Remarks
17.	Enter Email in the "Email" field	Email is entered.	Compulsory e.g.<sadc@sadc.com.my>
18.	Select Profile in the "Profile" field	Profile is selected.	e.g.<operator>
19.	<Start Block – add_wkr > Click on "Save"	"Manage Worker" page is displayed. <Stop Block – add_wkr >	
20.	Click on "Logout"	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	3. MFIT_AddClient	Test Script Version	1.0
-----------------	-------------------	---------------------	-----

Test Title	AddClient (Timer name : "add_clnt")
Test Objective	Add new client information into the system.
Pre-requisite	Valid username and password

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the "Username" field.	Username is entered.	Compulsory e.g. <mdec>
4.	Enter password in the "Password" field	Password is entered.	Compulsory e.g. <mdec01>
5.	Click on "General Data" tab	"General Data" page is displayed.	
6.	Click on "Companies and People" link	"Companies and People" page is displayed.	
7.	Click on "Add A New Client" link	"Create Worker" page is displayed.	
8.	Enter Known as in the "Known as" field	Known as is entered.	Compulsory e.g. <AYAMAS>
9.	Enter Legal name in the	Legal name is entered.	Recommended e.g.

Step	Description	Expected Result	Remarks
	"Legal name" field		<AYAMAS>
10.	Enter Company Registration no in the "Company Registration no" field	Company Registration no is entered.	Compulsory Unique e.g. <0001>
11.	Select Provider in the "Insert additional company profile" field	Provider is selected.	Compulsory
12.	<Start Block – add_clnt > Click on "Save"	"Manage Client" page is displayed. <Stop Block – add_clnt >	
13.	Click on "Logout"	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	4. MFIT_AddFinishedProd	Test Script Version	1.0
------------------------	--------------------------------	----------------------------	-----

Test Title	AddFinishedProd (Timer name : “add_fp”)
Test Objective	Adding finished product information into the system
Pre-requisite	Valid username and password

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the “Username” field.	Username is entered.	Compulsory e.g. <mdec>
4.	Enter password in the “Password” field	Password is entered.	Compulsory e.g. <mdec01>
5.	Click on “General Data” tab	“General Data” page is displayed.	
6.	Click on “Products” link	“Products” page is displayed.	
7.	Click on “Manage Finished Product” link	“Manage Finished Product” page is displayed.	
8.	Click on “Create Finished Product” button.	Create Finished Product page is displayed.	
9.	Enter Name in the “Name” field	Name is entered	Compulsory e.g. <Live Birds - Ross 2>
10.	<Start Block – add_fp> Click on “Save”	“Data stored Finished Product” page is displayed. <Start Block – add_fp>	
11.	Click on “Continue”	“Manage Finished Product” page is displayed.	
12.	Click on “Logout”	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	5. MFIT_BACKWARD TRACEABILITY	Test Script Version	1.0
-----------------	-------------------------------	---------------------	-----

Test Title	Backward Traceability (Timer name : "back_trace")
Test Objective	To test Backward Traceability function
Pre-requisite	Valid username and password

Step	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/mdtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the "Username" field.	Username is entered.	Compulsory e.g. <mdec>
4.	Enter password in the "Password" field	Password is entered.	Compulsory e.g. <mdec01>
5.	Click on "General Data" tab	"General Data" page is displayed.	
6.	Go to Home tab, Select Tracepoint	"Tracepoint" page is displayed.	
7.	Select "Search by reference or Tracepoint"	List of Tracepoint page is displayed	
8.	Select "Despatch of Live Birds"	"Despatch of Live Birds" page is displayed.	
9.	Click on " Search "	Search Results is displayed	
10.	Choose any Tracepoints	Tracepoint details is displayed	
11.	Click on "Backwards"	"Backwards" details is displayed	
12.	<Start Block – back_trace> Expand the "Backwards" button	List of backward tracepoint is displayed <Stop Block – back_trace>	
13.	Click on "Exit"	"Your Company" page is displayed.	
14.	Click on "Logout"	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test No.	Script	6. MFIT_Receiving and Stocking of Broiler DOC	Test Version	Script	1 . 0
-----------------	---------------	--	---------------------	---------------	-------------

Test Title	Receiving and Stocking of Broiler DOC (Timer name : "rcv_sdoc")
Test Objective	Receiving and Stocking
Pre-requisite	Valid username and password Delivery Order

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the "Username" field.	Username is entered.	e.g.<mdec>
4.	Enter password in the "Password" field	Password is entered.	e.g.<mdec01>
5.	Go to Home tab, Select Tracepoint-Broiler Farm Operation	Home page displayed.	
6.	Select Receiving and Stocking of Broiler DOC	Receiving and Stocking of Broiler DOC page displayed	
7 .	Select 'date of action'	Select date from calendar provided	Mandatory e.g.<29.07.07>
8 .	Select Supplier ID	Drop down list of Supplier id	<MDTCH>
9 .	Enter the Delivery order	Key in delivery order	Mandatory e.g.<060818H>
1 0 .	Select purchase product from drop down list.	Drop down list.	Mandatory e.g.<DOC Cobb>
1 1 .	Enter or Scan the Lot no of the broiler DOC received	Lot no appear if scanned	Mandatory e.g.<010101>
1 2 .	Enter Quantity of DOC received	Key in DOC	Mandatory e.g.<10,000>
1 3 .	Generate a new id for receiving & stocking by clicking "generate" button.	Code generated	e.g.< RG5MDTCFR000 0TB>
1 4 .	Select House no	Drop down list for House no	e.g.<Broiler House 99>
1	Enter stocking quantity	Key in amount	Mandatory

Step	Description	Expected Result	Remarks
5.			e.g.<3300>
16.	Enter total Dead On Arrival	Key in amount	Mandatory e.g.<0>
17.	<Start Block – rcv_sdoc > After complete, click the 'Save' button	The record is saved. <Stop Block – rcv_sdoc >	Mandatory
18.	Click on "Logout"	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	7. MFIT_Mortality Mobile Record	Test Script Version	1.0
Test Title	Mortality Mobile Record (Timer name : “mmr”)		
Test Objective	Mortality Mobile Record		
Pre-requisite	Valid username and password Total dead and cull birds		

S t e p	Description	Expected Result	Remarks
1	Launch IE 6.0 browser	IE 6.0 browser is launched	
2	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3	Enter username in the “Username” field.	Username is entered.	e.g.<mdec>
4	Enter password in the “Password” field	Password is entered.	e.g.<mdec01>
5	Go to Home tab, Select Tracepoint-Broiler Farm Operation	Home page displayed.	
6	Select Mortality Mobile Record	Mortality Mobile Record page displayed	
7	Select ‘date of action’	Select date from calendar provided	Mandatory e.g.<29.07.07>
8	Select the House no	List of House appear.	Mandatory e.g.< Broiler House 99>
9	Enter the total dead birds	Key in the amount.	Mandatory e.g.<10>
1 1	Enter number of cull birds	Key in the amount	Mandatory e.g.<5>
1 2	Enter Username and Password.	Key in username and password who carried out the process.	Mandatory
1 3	<Start Block – mmr > Click ‘save’ button	The record is saved <Stop Block – mmr >	Mandatory
1 4	Click on “Logout”	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	8. MFIT_Growth Monitoring Mobile Record	Test Script Version	1 . 0
-----------------	---	---------------------	-------------

Test Title	Growth Monitoring Mobile Record (Timer name : “gmmr”)
Test Objective	Growth Monitoring
Pre-requisite	Valid username and password

Step	Description	Expected Result	Remarks
1	Launch IE 6.0 browser	IE 6.0 browser is launched	
2	Type the URL link <https://secure2.foodreg.net/m dcf.html>	MFIT FoodReg page is displayed.	
3	Enter username in the “Username” field.	Username is entered.	e.g.<mdec>
4	Enter password in the “Password” field	Password is entered.	e.g.<mdec01>
5	Go to Home tab, Select Tracepoint-Broiler Farm Operation	Home page displayed.	
6	Select Growth Monitoring Mobile Record	Growth Monitoring Mobile Record page displayed	
7	Select ‘date of action’	Select date from calendar provided	Mandatory e.g.<29.07.07>
8	Select the House no	List of House appear.	Mandatory e.g.<house 99>
9	Enter the average body weight of birds	Key in the weight and the unit	Mandatory e.g.<0.05kg>
10	Enter Username and Password.	Key in username and password who carried out the process.	Mandatory
11	<Start Block – gmmr > Click ‘save’ button	The record is saved. <Stop Block – gmmr >	Mandatory
14	Click on “Logout”	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test Script No.	9. MFIT_Despatch of Live Birds	Test Script Version	1.0
Test Title	Despatch of Live Birds (Timer name : “desp_lb”)		
Test Objective	Despatching		
Pre-requisite	Valid username and password Customer		

S t e p	Description	Expected Result	Remarks
1	Launch IE 6.0 browser	IE 6.0 browser is launched	
2	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3	Enter username in the “Username” field.	Username is entered.	e.g.<mdec>
4	Enter password in the “Password” field	Password is entered.	e.g.<mdec01>
5	Go to Home tab, Select Tracepoint-Broiler Farm Operation	Home page displayed.	
6	Select Despatch of Live Birds	Despatch of Live Birds page displayed	
7	Select ‘date of action’	Select date from calendar provided	Mandatory e.g.<29.07.07>
8	Select Customer ID from the drop down list	List of Customer appear.	Mandatory e.g.<MDTCP>
9	Enter Delivery Order	Key in order amount.	Mandatory e.g.<060818H>
10	Select the House no. Click ‘Add’ button after complete. May add more than 1 lot receive.	Key in House no.	Mandatory e.g.<HOUSE 99>
11	Enter quantity of birds harvested.	Key in quantity.	Mandatory e.g.<2500>
12	Enter weight of birds harvested	Key in weight and unit.	Mandatory e.g.<4050kg>
13	Click generate button to generate Id code for the despatch	Code generated.	Mandatory
14	Select the medication withdrawal date from the date	Select date from calendar provided	e.g.<01.08.07>

Step	Description	Expected Result	Remarks
.	wizards.		
15.	Select the feed withdrawal date and time from the date wizards.	Select date and time from calendar provided	e.g.<17.08.07>
16.	<Start Block – desp_lb > Click ‘save’ button	The record is saved. <Stop Block – desp_lb >	Mandatory
17.	Click on “Logout”	User is successfully sign out and MFIT FoodReg main page is displayed.	

Test No.	Script	10. MFIT_Search by reference or trace point	Test Script Version	1. 0
-----------------	---------------	--	----------------------------	---------

Test Title	Search by reference or trace point (Timer name : “srch_tp”)
Test Objective	Search by reference or trace point
Pre-requisite	Valid username and password

S t e p	Description	Expected Result	Remarks
1.	Launch IE 6.0 browser	IE 6.0 browser is launched	
2.	Type the URL link <https://secure2.foodreg.net/m dtcf.html>	MFIT FoodReg page is displayed.	
3.	Enter username in the “Username” field.	Username is entered.	e.g.<mdec>
4.	Enter password in the “Password” field	Password is entered.	e.g.<mdec01>
5.	Go to Home tab, Select Tracepoint-Broiler Farm Operation	Home page displayed.	
6.	Select Search by reference or trace point	Search by reference or trace point page displayed	
7.	Go to Traceability Tab	Search by reference or trace point page appear.	
8.	Select the search criteria, by Reference, Product, Trace point, Date Start or Date End	Product and Trace point provide drop down list.	e.g.<Product = old- chick ross>
9.	Key in search criteria	Search criteria is entered	
10.	< Start Block – srch_tp > Click Search button	Search Result appears. < Start Block – srch_tp >	
11.	Click on “Logout”	User is successfully sign out and MFIT FoodReg main page is displayed.	

Appendix D: SUS DATA

Appendix D: SUS DATA

User	Q1		Q2		Q3		Q4		Q5		Q6		Q7		Q8		Q9		Q10	
	Scale Position	Score																		
1	5	4	1	4	4	3	1	4	4	3	2	3	5	4	5	0	4	3	1	4
2	5	4	1	4	4	3	2	3	5	4	1	4	5	4	1	4	5	4	2	3
3	3	2	1	4	4	3	4	1	5	4	1	4	4	3	2	3	4	3	2	3
4	4	3	1	4	5	4	1	4	5	4	1	4	5	4	1	4	5	4	2	3
5	5	4	1	4	5	4	1	4	5	4	1	4	5	4	1	4	5	4	1	4
6	4	3	2	3	3	2	2	3	3	2	2	3	3	2	2	3	4	3	1	4
7	3	2	4	1	4	3	3	2	4	3	2	3	3	2	4	1	3	2	4	1
8	5	4	1	4	5	4	1	4	4	3	2	3	4	3	1	4	5	4	1	4
9	4	3	3	2	5	4	1	4	3	2	1	4	5	4	1	4	4	3	2	3
10	3	2	2	3	2	1	2	3	4	3	1	4	1	0	2	3	1	0	1	4
11	5	4	1	4	4	3	2	3	4	3	2	3	5	4	4	1	5	4	1	4
12	3	2	2	3	4	3	1	4	4	3	3	2	4	3	4	1	4	3	2	3
13	3	2	3	2	2	1	4	1	4	3	3	2	1	0	3	2	3	2	3	2
14	2	1	4	1	3	2	5	0	3	2	4	1	2	1	2	3	3	2	4	1
15	5	4	2	3	4	3	1	4	4	3	3	2	5	4	2	3	5	4	1	4
16	4	3	1	4	5	4	1	4	4	3	1	4	5	4	2	3	4	3	2	3
17	5	4	1	4	4	3	2	3	4	3	1	4	5	4	3	2	4	3	2	3
18	5	4	2	3	5	4	1	4	4	3	3	2	4	3	2	3	3	2	1	4
19	4	3	1	4	4	3	2	3	4	3	1	4	5	4	1	4	4	3	2	3
20	5	4	2	3	5	4	1	4	4	3	3	2	4	3	1	4	4	3	1	4
21	3	2	2	3	4	3	2	3	4	3	1	4	4	3	1	4	4	3	2	3

