ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

A SEMANTIC BASED SEARCH FOR TEST CASE REPOSITORY

Shivakumar Swamy N[1*], Sanjeev C. Lingareddy[2]
[1*]Ph.D Scholar, Dept. of CSE ,JJTU,Jhunjhunu,Rajastan-333001
[2]Prof. and Head, Dept. of CSE Alpha College of Engineering, Bangalore
Correspondence Author*:* ns.shivakumar@gmail.com

Keywords: Test Driven Development, RDF, Intelligent Systems, Coverage Contribution and Ontology

Abstract
Semantic Web [1] is a new generation web which emphasize on the effective utilization of Meta data as semantic/relevant information of a resource to derive its context. In semantic web the meta data is represented as RDF (Resource Description Framework) [2-3] and every object in RDF is conceptualized using the Ontology. The ontology describes the concepts and the relationship between the concepts. In this paper a new solution is proposed which uses the semantic information of test cases to search for efficient unit test, test case and test suite from the huge repository of test cases. The semantic information for every test case is represented using RDF describing the coverage information for every test case. The coverage analysis is considered as important criteria for ranking the unit test, test case and test suite. The search results can be filtered by with the explicit specification of coverage criteria.

Introduction
Semantic Intelligence defines the power of unstructured data analysis by identifying the required semantics [4-6]. The intelligent systems are built and help in decision making process. Especially semantic analysis required in Industries to analyze the Business data. In the world of semantic web semantic analysis is performed on the web data. The web has disjoint set of objects in both text and multimedia space. Semantics can be applied to any type of information. In WWW the semantics are embedded to web pages by the author of web pages to define the context of the information. The tools like a search engine [7] utilizes these embedded semantics to provide context aware information for the user query. Semantics can be applied in various domains like Knowledge Management [8], Decision making process [9] and Corporate Intelligence.
The basic elements of Semantic Web are *Ontology [10-12] and RDF*. Ontology science deals with the concepts and relations for a specific domain. The concept wraps the properties and behavior to describe the semantics. Ontology provides three basic types of relations such as Object Property, Data Property and Functional Property. The Object Property [13] relates two object resources. Data Property [13] relates object to string literal for example <John, hasAddress, "Bangalore city"> the hasAddress relates the person to string literal describing the address of person *John*. Functional Property [13] is a variant of Data property where it relates Object to unique string literal which is not reused across objects. The second vital element of Semantic Web is Resource Description Framework -RDF which uses the ontology concepts and relations to describe required semantics. The RDF semantics is a collection triplets component. Every triplet has domain and range as objects mapped to ontology concepts. RDF can use those properties defined in Ontology to establish triplet. The RDF is generally embedded in resource and a tool operates on this captured semantics to become intelligent.
Motivation – Current Semantic web architecture is an evident for context driven information search which motivates to use semantics for analysis of diverse information. The RDF enables to describe meta data as structured data and hence it can be embedded easily to a resource. The ontology and RDF is domain specific but the tools operates on this meta data is generic and hence RDF in conjunction with Ontology describes meaning full context for the resource. In current IT infrastructure especially when Test driven development [14-18] is used to create automated test cases for qualifying the products creates a challenging environment to describe the efficiency of test cases inspire of having coverage reports and hence a guiding system is required for quality professionals to identify testing quality.
Contribution – In current solution an intelligent tool is prototyped for identifying the efficient test cases. One such parameter which describes the efficiency is the functionality coverage by test cases. From the coverage reports only the coverage analysis of code is performed but which test cases or unit test contributed for coverage is not recorded. The current solution emphasizes on deriving the efficiency for test cases based on their coverage contribution. The coverage contribution for every test case and unit tests are recorded as RDF semantics which is embedded for a test case using customized annotations. Usage of semantics for describing test cases efficiency enables the quality professionals to take decision on the quality of test cases. The current solution takes coverage factor probability value as input and returns unit tests and test cases without ranking the results.

Related work
Test driven development has been one of the major growing areas in the area of testing. There exist many studies to estimate the

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

quality achieved with the test driven development.
George and Williams [19] performed experiments involving 24 pairs of professional programmers. One group developed a small JAVA program by applying test driven development, whereas the other (control) group used the waterfall lifecycle model (a classical software engineering development model). Farmer approach produced better quality code compared to the later. Williams et al. [20] carried out a case study in IBM, shows that the test driven development had 40% fewer defects when compared with the code of an experienced team using an ad-hoc testing approach. Edwards proposes the use of test driven development as a testing practice in a classroom as a pilot study [21]. Students using test driven development made 45% fewer mistakes than those who did not use them. Pancur et al. investigated the differences between test driven development and normal testing approach [22]. The results show that test driven development had better efficiency than normal testing procedure.
A controlled experiment was conducted with 14 voluntary industrial participants [23] in Canada. Half of the participants used a test-first practice, and half of these used a test-last practice to develop two small applications that took 90-100 minutes, on average, to complete. The research indicated little to no differences in productivity between the methods, but that test-first may induce developers to create more tests and to execute them more frequently. Another controlled experiment was conducted with 28 practitioners at the Soluziona Software Factory in Spain [24]. Each practitioner completed one programming task using the test driven development practice and one task using a test-last practice, each taking approximately five hours. Their research indicated that test driven development requires more development time, but that the improved quality could offset this initial increase in development time. Additionally TDD leads developers to design more precise and accurate test cases.

Problem definition

In large IT companies, Test Driven Development (TDD) is a well-known framework used to perform rigorous testing and the main problem is in identifying the test cases based on their efficiency. So the coverage factor formulates the parameter for defining the efficiency. Even if the coverage report for the functional code is available, it is impossible from the reports to explore the test cases contribution for the coverage. This solution finds the efficiency of the test cases and unit tests bases on their contribution for the functional coverage criteria. The functional coverage reports are gathered and analysis is performed to define probabilistic model for every test case and unit test to define their efficiency. It is assumed that the testers upload the compiled test cases to our framework and also it is assumed that the testers have annotated the unit tests and test cases with the custom annotations defined by this framework.

Computational model

Table 1: Basic Notations

Si.No.	Notation	Meaning
1	τ	Coverage Factor
2	ρ_u	Line coverage of unit tests
3	ρ_t	Line coverage of test case
4	σ_u	Branch Coverage of unit tests
5	σ_t	Branch Coverage of test case
6	$\bar{U}$	Vector of Unit Tests
7	$\bar{T}$	Vector of Test Cases
8	$\bar{S}$	Vector of test suites
9	$\overline{NL}$	Vector of lines count indexed according to the corresponding classes
10	$\overline{FM}$	Vector of Functional Methods indexed according to the corresponding classes
11	$\overline{FC}$	Vector of Functional Classes
12	$\bar{M}$	Vector of functional methods.

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

Definitions:

Test suite vector

Test suite Vector is a list of all test suites present in repository, where every test suite is a super set or collection of test cases and is defined as,

$$S_i = \{S_i \supseteq x : x \subseteq \bar{T} \; and \; |x| >= 1; 1 \leq i \leq |S|\} \quad (1)$$

Test case vector

Test case Vector is a list of all test cases present in repository, where every test case is a superset of unit tests and may belongs to some test suite and is defined as,

$$T_i = \{T_i \supseteq x : x \subseteq \bar{U} \; and \; |x| >= 1 \; or \; T_i \in \bar{S}; 1 \leq i \leq |T|\} \quad (2)$$

Unit test vector

Unit test vector is a list of all unit tests available in repository, where every unit test should be a part of some test cases and is defined as,

$$U_i = \{U_i \in t : \exists t \in \bar{T} \; \& \; 1 \leq i \leq |U|, \} \quad (3)$$

Lines count and branch count vector

Lines count vector is a map of functional class to its corresponding lines count and is defined as

$$NL_i = NB_i = \{< c_i, n >: n \in Z^+ \; \& \; c_i \in \overline{FC}; 1 \leq i \leq |FC|\} \quad (4)$$

Where, the line count of specific i[th] class can be accessed using $NL_i.c_i$ and $NB_i.c_i$. Similarly method lines count vector is a map of method to its lines count and is defined as,

$$ML_i = \{< m_i, n >: n \in Z^+ \; \& \; m_i \in \bar{M}; 1 \leq i \leq |M|\} \quad (5)$$

Functional method vector

Similar to the lines count vector Functional method vector is a map of methods vector to its corresponding class and is defined as,

$$FM_i = \{< c_i, \overline{mt_i} >: c_i \in \overline{FC} \; \& \; \overline{mt_i} \subseteq \bar{M}; 1 \leq i \leq |FC|\} \quad (6)$$

Where, any i[th] class functional methods can be accessed using FM_i which returns the vector of methods subset.

Coverage dependency

The coverage dependency is defined between the unit test and functional method. If unit test say 'u' invokes a functional method 'm' then unit test depends on functional method for coverage and indicated as u→m, where u ϵ $\bar{U}$ and m ϵ $\bar{M}$. The line coverage dependency is the number of lines out of total number of lines executed based on the invocation from the respective unit test. The line coverage dependency between the unit test u ϵ $\bar{U}$ and m ϵ $\bar{M}$ is represented as $l_{u,m}$.

Similarly the branch coverage dependency is the number of branches covered out of total number of branches based on the invocation done by the corresponding unit test and the branch coverage dependency between the unit test u ϵ $\bar{U}$ and m ϵ $\bar{M}$ is represented as $b_{u,m}$.

Line coverage

Line coverage is probability of number of lines invoked to total number of lines in class. Line coverage of unit test (ρ_u) is the probability that total number of lines covered by the invocation of corresponding functional method to total number of lines in class. Line coverage of Test case (ρ_t) is the average of all its unit tests line coverage.

The line coverage of i[th] unit test is defined as,

$$\rho_{u_i} = lu_{i,}m \; / \; NL.c \; if \; m \in FM.c \quad (7)$$

The line coverage of the test case is summation of line coverage of all the unit tests to the total number of unit tests of that test case and defined as,

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

$$\rho_{T_i} = \frac{\sum_{j=0}^{|FM.T|} \rho_{u_j}}{|FM.T|} \quad (8)$$

Similarly the line coverage of test suite is the summation of line of all the test cases to the total number of test cases belonging to the test suite and any i[th] test suite line coverage is defined as,

$$\rho_{S_i} = \frac{\sum_{j=0}^{|S_i|} \rho_{T_j}}{|S_i|} \quad (9)$$

Branch coverage

Branch coverage is probability of number of branches invoked to total number of branches in class. Branch coverage of unit test(σ_u) is the probability that total number of branches covered by the invocation of corresponding functional method to total number of branches in class. Branch coverage of Test case (σ_t) is the average of all its unit tests branch coverage.

The branch coverage of i[th] unit test is defined as,

$$\sigma_{u_i} = bu_{i,}\, m / NB.c \; if \; m \in FM.c \quad (10)$$

The branch coverage of the test case is summation of branch coverage of all the unit tests to the total number of unit tests of that test case and defined as,

$$\sigma_{T_i} = \frac{\sum_{j=0}^{|FM.T|} \sigma_{u_j}}{|FM.T|} \quad (11)$$

Similarly the branch coverage of test suite is the summation of branch of all the test cases to the total number of test cases belonging to the test suite and any i[th] test suite branch coverage is defined as,

$$\sigma_{S_i} = \frac{\sum_{j=0}^{|S_i|} \sigma_{S_i}}{|S_i|} \quad (12)$$

Coverage factor

The coverage factor is the average of line and branch coverage. The coverage factor for a unit test is its summation of line and branch coverage by 2 and defined as,

$$\tau_{u_j} = \frac{\rho_{u_j} + \sigma_{u_j}}{2} \quad (13)$$

The coverage factor for test case is the summation of average coverage factor of all unit tests and is defined as,

$$\tau_{T_i} = \frac{\sum_{j=0}^{|FM.T|} \tau_{u_j}}{|FM.T|} \quad (14)$$

Similarly the coverage factor for test suite is the summation of average coverage factor of all its test cases and is defined as,

$$\tau_{S_i} = \frac{\sum_{j=0}^{|FM.T|} \tau_{T_j}}{|FM.T|} \quad (15)$$

System architecture

The solution framework defines the ontology model for RDF semantics to adhere. The RDF semantics uses the concepts and properties defined to create a Meta data for test cases. In Semantic Web world the systems behave intelligently by understanding the semantics of the resources.

Ontology model for the test frame work

Ontology provides the conceptual insight for applications belonging to the specific domain and hence the ontology is domain specific but not generic. Every domain has its own ontology defined, for example ontology for employee domain in IT industry is different from the employee domain of Academic institutions. The search engine utilizes the embedded Ontology (e-Ontology) to

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

arrive at the domain of the resource. For every resource RDF is embedded e-RDF adhering to ontology to describe the context or the role of resource in a particular domain. The resource can be web documents (web search),text documents(Natural Language Processing) so on and so forth. In Test frame work the test case is the main resource. The following are the concepts of the Test Frame Work Ontology Model.

Unit test

This component is the smallest unit of frame work where the required assertions are made. The unit test invokes the required functionality to check its correctness with one or more assertions. The unit test is described with three important properties, belongsTo is the relationship between Unit Test and Test Case defining the presence of unit test with in test case. hasLineCoverage and hasBranchCoverae is defines the probability of lines and branches covered by the unit tests.

Test case

The test case is an aggregation unit tests. This ensures that all the required tests are executed in one time to validate the overall functionality. The test case is embedded with the required RDF describing the coverage information of the test case. Similar to the unit tests even test cases are associated with line and branch coverage properties

Test suite

The test suite is an aggregation of test cases. This is to run the test cases of all features of product in totality. The test suite can have any number of test suites as components.

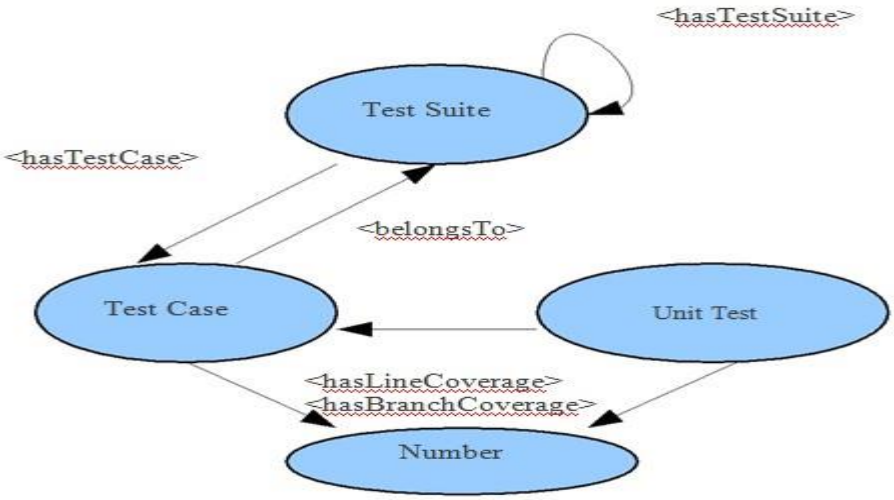

Figure 1: Ontology Model for Test Frame Work

RDF generator

This is a vital component responsible for generating required semantics for the test cases serialized as RDF. The RDF is adhered to the test frame work ontology model. The RDF generator uses the coverage measurement of test cases to extract the line and branch (condition) coverage information as semantic information. The RDF information of every test case is stored as set of resource files forming *RDF repository*. The coverage reports provide the coverage measurement in XML format adhering to specific structure. The open source tool *cobertura* is used to perform coverage analysis of test cases. The RDF is embedded in to the test cases using custom annotations. The RDF generator takes Test case files, Coverage reports and Ontology Model as inputs to generate the RDF.

RDF crawler

The crawler is the key component of search engine which is responsible for crawling the WWW to collect web pages and performs the page segmentation. The RDF crawler is the customized crawler that grabs the test cases from the test case repository and fetches the location of corresponding RDF. The crawler process the RDF and serializes the processed information to structured data model as RDBMS schema. The RDBMS has required schema for storing the serialized RDF.

Test searcher

The test searcher takes the user input and performs analysis on the processed semantics to identify the efficient unit test, test case and test suite. The user gives the minimum coverage criteria as input. The test searcher performs the analysis on the processed semantics to fetch the test cases sorted in descending order according to the coverage measurement. The RDF generator and crawler components are scheduled to run as cron jobs in background where as Test searcher component is invoked when user makes a request.

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

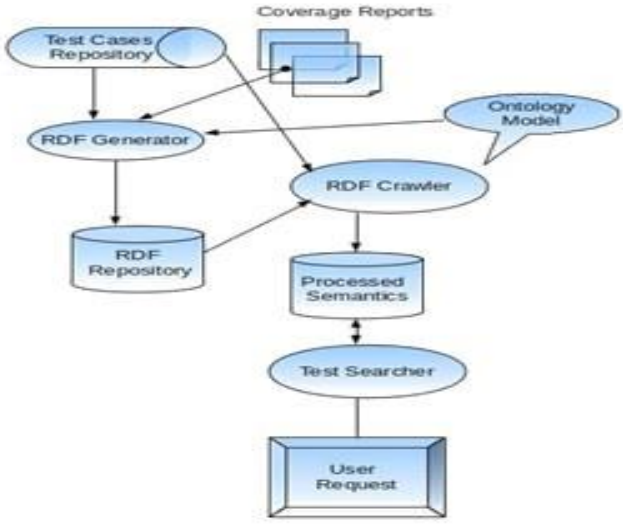

Figure 2: Architecture of PETWS

Algorithms

The current solution has two important components RDF Generator and RDF Crawler. The test cases provides information about every unit test and test case as itself. The information in test cases is embedded using the annotations defined by this framework. The RDF Generator utilizes this piece of annotation data to generate appropriate semantics/RDF. The RDF is generated for every test case adhering to the PETWS ontology.

There are two main annotations, @PETWSTestCaseData which gives information about name of test case, test suite to which it belongs to and @PETWSUnitTestData gives the information about test method name, the corresponding functional method it invokes. The RDF generator uses this abstract meta data to generate more detail meta data (RDF) form every test case as per the algorithm in table 2.

Table 2: PETWS RDF Generator

Input:

Cr- Coverage Report.

$\overline{T}$- Set of test cases.

Output:

$\overline{T - RDF}$- Set of RDF for every Test case.

Process:

for-each test-case $\epsilon\, \overline{T}$
do
$\overline{UT}$ < - test-case.unitTests ; where $\overline{UT} \subseteq \bar{T}$
RDF-Triplets <- ϕ
for each unit-test $\epsilon\, \overline{UT}$
do
if
unit-test.hasAnnotation(@PETWSUnitTestData)
then
#get the functional method invoked.
m <- unit-test.@PETWSUnitTestData.method;
#get the number of lines invoked from coverage report.
$l_{unit\text{-}test,m}$ < -m.Cr.lines;
$b_{unit\text{-}test,m}$ < -m.Cr.branches;
calculate $\rho_{unit\text{-}test}$,$\sigma_{unit\text{-}test}$ as per the equations (6) and (9).
RDF-triplet1 <- <unit-test,hasLineCoverage, $\rho_{unit\text{-}test}$ >
RDF-triplet2<-<unit-test,hasBranchCoverage, $\sigma_{unit\text{-}test}$>

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

RDF-triplets <- RDF-triplets ∪ RDF-triplet1 ∪ RDF-triplet2.
end if
done
Calculate line coverage and branch coverage $\rho_{test\text{-}case}$ $\sigma_{test\text{-}case}$ of
test case as per the equations (7) and (10).
Serialize RDF-triplets to RDF-File. T-RDF <- T-RDF ∪ RDF-triplets
Done

The RDF Generator iterates through every unit test to extract the abstract semantic embedded in the form of annotation and it process the coverage report to get the number of lines and branches executed based on the invocation performed by the corresponding test method. The RDF generator also calculates the line coverage and branch coverage probability values; this piece of information forms the detail meta data or required semantics to generate the RDF triplets. The triplets are serialized as XML and saved to the repository. The RDF file is generated for every test case. The basic operation here is to generate RDF-triplets and hence this has complexity of $O(n^2)$.
The RDF Crawler component crawls through the RDF Repository to process the semantics and serialize data into structured RDBMS tables. The main job of the crawler is to compute the coverage factor from line and branch coverage information available as semantics through RDF. The coverage factor in a way reflects the efficiency of unit tests and test cases. The test searcher takes the probability value provided by user to get the unit tests, test cases having coverage factor at least the coverage factor provided by the user. The RDF crawler has the process explained as per the algorithm given in table 3.

Table 3: PETWS RDF Crawler

Input:
$\overline{T-RDF}$- Set of Test cases RDF.
Output:
Coverage factors, τ_{u_i} τ_{T_i}
Process:
for each t-rdf ∈ $\overline{T-RDF}$
do
for each triplet ∈ t-rdf
do
get line and branch coverage for unit tests.
ρ_u = triplet.line Coverage.
σ_u = tipltet.brachCoverage.
calculate the coverage factor for unit test 'u' τ_u.
done
calculate the coverage factor τ_T for test case with t-rdf as semantics.
Serialize the t-rdf to repository and also dump coverage factor for every test case
and unit test.
Done

The algorithm runs on every Test case RDF and iterates over every unit test triplet to get line and branch coverage information. Calculation of coverage factor forms the basic operation, if there are *n* test cases and *m* triplets then the algorithm has the complexity of O(m*n). The RDF generator utilizes the xml coverage report generated by the cobertura. The key difference between the cobertura and PETWS is that the cobertura gives the number of lines in method executed based on the invocation. PETWS utilizes this number to arrive at the probability of line and branch coverage done by the test cases. The PETWS boils down even at unit test method level to rank them based on the cobertura coverage report.

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

Experimental results

The experimental results orchestrate the unit test, test cases efficiencies based on the coverage contribution. The entire solution is implemented as a search engine, where the crawler is set up to run as cron job, the RDF generator also scheduled for event based execution, the event when the users uploads the test classes with required annotations the database is updated by the RDF generator to add the semantics of new uploaded test cases. For demonstration purpose the custom test cases are written with proper annotations for every test method. The coverage report is also grabbed by the search engine by the users and it is serialized as XML[19] file. The RDF generator process the coverage report to calculate the required branch and line coverage for test cases. The result is serialized as RDF generator. The RDF generator and Crawler uses the JAVA Jena Api [20-22] for RDF generation and its parsing, MySQL, Tomcat and eclipse completes the experimental setup. As shown in figure 1 the unit tests are plotted across line and branch coverages forming overall coverage factors. Among seven unit tests the unit test T6 has maximum line coverage as compared to rest six unit tests. The behavior of test case T4 is same for both line and branch coverage. This indicates that the functionality piece tested by T4 has more branches but less number of lines to cover as compared to T6. In Industry the quality professionals usually defines the benchmark for the coverage, for benchmark 0.5 probability, T3 and T5 goes out of scope and considered as low quality unit tests.

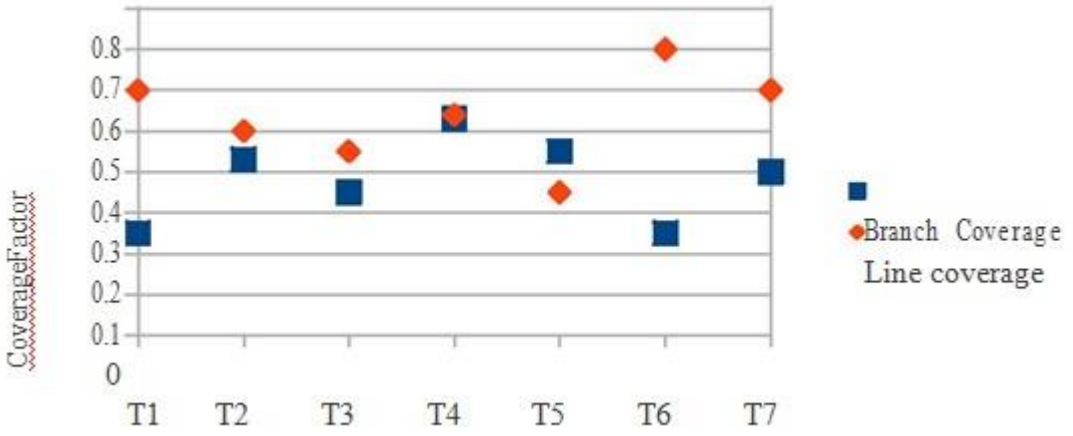

Figure 3: Unit tests Line vs Branch coverage

The PETWS search results in list of unit tests and test cases based on the coverage factor input provided by the user. The coverage factor input is average of branch and line coverage contributions. The stacked area chart in figure 3 provides the evident that number of test elements (unit tests and test cases) retrieved for a specific coverage input. The prefix 'p' in x-axis indicates the probability values for example 10P is 0.1 probability. There are 5 unit tests and 3 test cases are returned for 10% coverage factor and hence totally it contributes to 8 test elements. The highest number of test elements are retrieved for 30% coverage factor i.e 18 and least number of test elements are retrieved for 70% coverage factor, the number of test elements shrinks when coverage factor is moved from lower to higher values. Therefore one can infer that the test elements are poor quality from coverage perspective
.

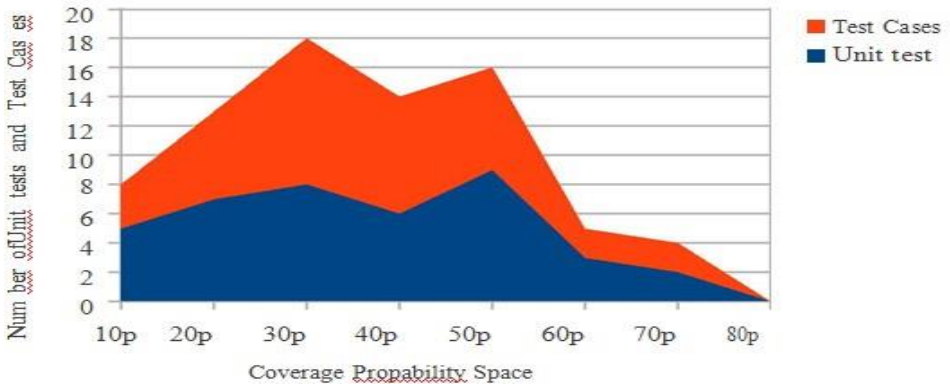

Figure 4: Number of Test elements vs Coverage Criteria

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

As described the current solution relies on the Meta Data embedded for test cases. The semantic annotation or Meta data formatted to RDF has triples describing the line and branch coverage factors. The RDF generator runs through every test case and creates single RDF which has 'hasLineCoverage' and 'hasBranchCoverage' triples for describing the coverage factors. One additional property 'belongsTo' indicates the presence of unit test inside test case. The graph fig 3 is plotted on Test Case vs Coverage triplets. The RDF generator produces twice the number of triplets as compared to <belongsTo>. The test case TC4 produces maximum number of coverage triplets 12 for 6 unit tests and totally 18 triplets are generated by RDF triplets. If 'n' <belongsTo> triplets exists, then n+2n triplets are generated and this is generated for all test cases. Therefore RDF triplets is vital and expensive components.

The sample RDF file generated by RDF generator is depicted in figure 2. As shown in figure 2 the testDescending has branch coverage of 0.5 and belongs to DesAscTest test case. The instanceOf object property defines the type of test element. The DesAscTest test element has instanceOf set to TestCase, which identifies it as test case

```
<rdf:RDF>
  <rdf:Description rdf:about="http://www.bang.maans3.com/testDescending">
    <j.0:hasBranchCoverage>http://www.bang.maans3.com/0.5</j.0:hasBranchCoverage>
    <j.0:hasLineCoverage>http://www.bang.maans3.com/0.47058823529411764</j.0:hasLineCoverage>
    <j.0:belongsTo>DesAscTest</j.0:belongsTo>
    <j.0:instanceOf>http://www.maanscube.com/PETWS/UnitTest</j.0:instanceOf>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.bang.maans3.com/testAscending">
    <j.0:hasBranchCoverage>http://www.bang.maans3.com/0.5</j.0:hasBranchCoverage>
    <j.0:hasLineCoverage>http://www.bang.maans3.com/0.47058823529411764</j.0:hasLineCoverage>
    <j.0:belongsTo>DesAscTest</j.0:belongsTo>
    <j.0:instanceOf>http://www.maanscube.com/PETWS/UnitTest</j.0:instanceOf>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.bang.maans3.com/DesAscTest">
    <j.0:hasBranchCoverage>http://www.bang.maans3.com/0.5</j.0:hasBranchCoverage>
    <j.0:hasLineCoverage>http://www.bang.maans3.com/0.47058823529411764</j.0:hasLineCoverage>
    <j.0:instanceOf>http://www.maanscube.com/PETWS/TestCase</j.0:instanceOf>
    <j.0:hasID>
      http://www.bang.maans3.com/0542ed53-1b8c-4518-8eaf-9a56c5d92f54
    </j.0:hasID>
    <j.0:presentIn>http://www.bang.maans3.com/TestSuiteFirst</j.0:presentIn>
  </rdf:Description>
</rdf:RDF>
```

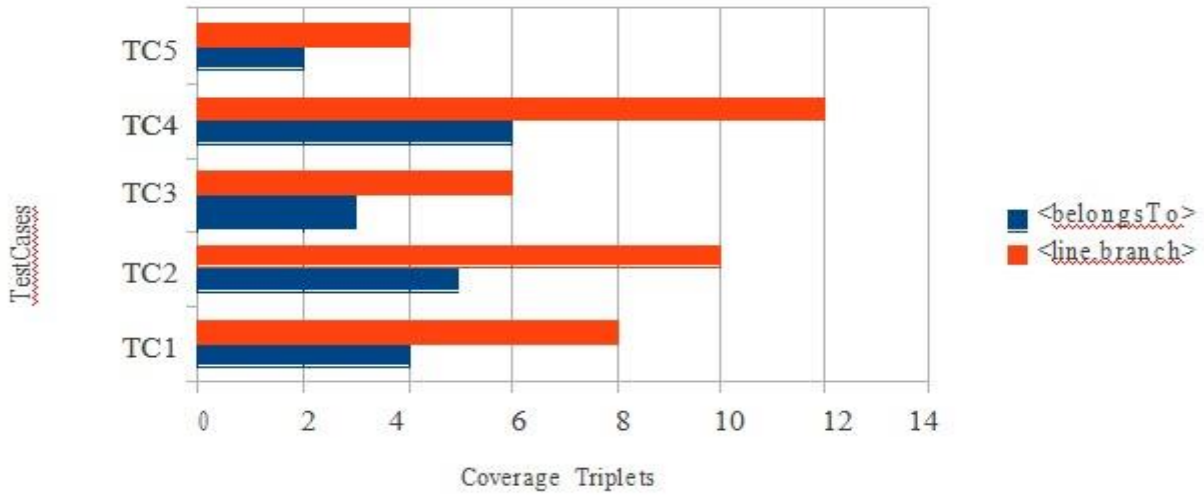

Figure 5: Test Cases vs Coverage Triplets

Conclusion

The semantics defines the holistic data about a resource and which enables to develop the intelligent systems for analysis purpose. The current solution uses this semantics to annotate the test cases and unit tests to derive the efficiency. The coverage contribution of test cases forms the fact for deriving the efficiencies and they are defined as probability values which is serialized to RDF semantics. The test searcher perform the analysis on RDF to produce results. From the experimental results it is evident that coverage contribution is a trustful fact for deriving the efficiencies and the usage of semantics to capture such information helps

ISSN: 2349- 5197
Impact Factor (PIF): 2.138

the tools to operate for application specific activities. This solution can be applied to the big repository of test cases in IT industries to analyze their efficiency. Knowing the efficiency helps to perform accurate testing. Current solution does not define any ranking methodology for rating tests cases based on their efficiency. This solution can be enhanced to define the ranking model by considering the relationships between the unit tests.

References

1. http://en.wikipedia.org/wiki/Semantic_Intelligence
2. Kamath S, Piraviperumal D, Meena G, Karkidholi S and Kumar, K. “A semantic search engine for answering domain specific user queries”. IEEE transactions, pp 1097-1101.
3. Harth, Andreas, and Stefan Decker. "Optimized index structures for querying RDF from the web." Web Congress, 2005. LA-WEB 2005. Third Latin American. IEEE, 2005.
4. Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." Scientific american 284.5 (2001): 28-37.
5. Horrocks, Ian, et al. "SWRL: A semantic web rule language combining OWL and RuleML." W3C Member submission 21 (2004): 79.
6. Ankolekar, Anupriya, et al. "DAML-S: Web service description for the semantic web." The Semantic Web—ISWC 2002. Springer Berlin Heidelberg, 2002. 348-363.
7. Calsavara, Alcides, and Glauco Schmidt. "Semantic search engines." Advanced Distributed Systems. Springer Berlin Heidelberg, 2004. 145-157.
8. Alavi, Maryam, and Dorothy E. Leidner. "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues." MIS quarterly (2001): 107-136.
9. Janis, Irving L., and Leon Mann. Decision making: A psychological analysis of conflict, choice, and commitment. Free Press, 1977.
10. McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." W3C recommendation 10.10 (2004): 2004.
11. Gomez-Perez, Asuncion, Mariano Fernández-López, and Oscar Corcho-Garcia. "Ontological engineering." Computing Reviews 45.8 (2004): 478-479.
12. Klein, Michel. "Interpreting XML documents via an RDF schema ontology." Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on. IEEE, 2002.
13. Motik, Boris, et al. "Owl 2 web ontology language: Profiles." W3C recommendation 27 (2009): 61.
14. Beck, Kent. Test-driven development: by example. Addison-Wesley Professional, 2003.
15. Janzen, David, and Hossein Saiedian. "Test-driven development: Concepts, taxonomy, and future direction." Computer 9 (2005): 43-50.
16. Venners, Bill. "Test-driven development." A Conversation with Martin Fowler, Part V (Cited 1 Apr 2009). URL http://www. artima. com/intv/testdrivenP. html (2002).
17. Williams, Laurie, E. Michael Maximilien, and Mladen Vouk. "Test-driven development as a defect-reduction practice." Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on. IEEE, 2003.
18. Geras, Adam, M. Smith, and James Miller. "A prototype empirical evaluation of test driven development." Software Metrics, 2004. Proceedings. 10th International Symposium on. IEEE, 2004.
19. George, B. and Williams, L. A structured experiment of testdriven development. Information and Software Technology 46 (May 2004), pp.337–342
20. Williams L., Maximilien, E., and Vouk, M. Test-driven development as a defect-reduction practice. In Proc. of the 14th IEEE Int’l Symposium on Software Reliability Engineering (ISSRE’03), (Denver, Colorado, USA, 2003), IEEE CS Press, pp.34-48.
21. Edwards, S. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In Proc. of the Int’l Conference on Education and Information Systems: Technologies and Applications (EISTA’03), (Orlando, Florida, USA, 2003).
22. Pankur M., Ciglaric M., Trampus M. and Vidmar T.Towards empirical evaluation of test-driven development in a university environment. In EUROCON 2003. Computer as a Tool. The IEEE Region 8 Volume: 2, (Ljublijana, Slovenia, 2003), IEEE CS Press, pp.83, 86.
23. A. Geras, M. Smith, and J. Miller, "A Prototype Empirical Evaluation of Test Driven Development," in International Symposium on Software Metrics (METRICS), Chicago, IL, 2004, pp. 405 -416
24. G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C.A. Visaggio, "Evaluating Advantages of Test Driven Development: a Controlled Ex periment with Professionals," in International Symposium on Empirical Software Engineering (ISESE) 2006, Rio de Jaiero, Brazil, 2006, pp.364-371