Test Size Estimation for Object Oriented Software Based on Analysis Model

Chamundeswari Arumugam*, Chitra Babu
Department of Computer Science and Engineering, Sri Siva Subramaniya Nadar College of Engineering, Rajiv Gandhi Salai (OMR), SSN Nagar, Tamil Nadu, India, 603110.

* Corresponding author. Email: chamundeswaria@ssn.edu.in.
Manuscript submitted August 29, 2014; accepted May 12, 2015.
doi: 10.17706/jsw.10.6.713-729

Abstract: Software test size estimation at the early analysis phase of software development lifecycle is crucial for predicting the associated effort and cost. This paper proposes a new method namely System Test Size Point (STSP) to estimate the system testing size of the object oriented software. The novel approach for the system test size estimation at the analysis phase is based on the Use Case Model (UCM) which adapts the Function Point Analysis (FPA) technique. The various features such as use case graph, test set, test set suite, relationships, main flow and alternate flow, are extracted from the UCM. Based on this, UCM components for estimation are derived. The FPA components are appropriately mapped to the UCM components and the complexity based on the weightage is specified to calculate the STSP. This proposed test size estimation approach has been evaluated with the object oriented software developed in our software engineering laboratory to assess its ability to predict the system test size.

Key words: Function point, object-oriented software, system testing size estimation, use case model.

1. Introduction

Software estimation is essential in every software organization for effective decision making. Both development and testing impact the size of an object oriented software system. Although several models exist for estimating the development size of object oriented software[1], very few works focus towards the estimation of testing size. Testing is a challengeable task, which requires quite a lot of effort to deliver high quality software. The size and effort estimate for testing must be good enough to provide sufficient time to conduct tests and guarantee that the product enters the market adequately tested, avoiding negative reactions among consumers and damages in the image of the company responsible for the software. Software organizations in fact assign a test manager exclusively to take the responsibility of estimating the testing size and accordingly plan the software testing management activities.

FPA [2] is a popular method for software size estimation. This method estimates the software size based on the functionality from the requirements specification, independent of the technology used to build the software system. FPA was proposed by International Function Point User Group (IFPUG) and it has become a standard for size estimation. FPA cannot be directly applied to Object Oriented (OO) software due to the mismatch in the features supported by it. However, this method has been suitably adapted for estimating size of the OO software [1]. The FPA component was mapped to use case model in an earlier work [1] to estimate the size of OO software during the analysis phase itself. While this approach facilitated the

estimation of development size of OO software, it did not estimate the testing size. System testing in a typical software life cycle is a phase where the software system is tested in its entirety to verify and validate the requirements. It is important to estimate the effort involved in the various activities related to system testing at the early stage of the software life cycle.

Table 1. Overview of the Existing Test Estimation Methods

Author(s)	Estimation Method Name	Data Used	Adapted
Almeida *et al*	Test activity effort	Use case	Nageswaran method
Aranha *et al*.	Test execution effort	Test specification	-
Ashish *et al*.	Test effort	Software requirements specification	-
Baudry *et al*.	Test effort	Class diagram	-
Kushwaha *et al*.	Test effort	Source code	-
Nageswaran	Test effort	Use case	Use case point
Thomas mccabe	Cyclomatic complexity	Source code	-
Veenendaal *et al*.	Test point	Functional requirements	-
Xiaochun *et al*.	Test execution effort	Use case	-
Zhou *et al*.	Test suite size	Use case	-

UCM generally provides the complete information about all the functionality, addressed by any given software system. It also has certain distinct advantages in capturing the system requirements earlier, in the software life cycle [3]. Further, it captures different granularities such as brief, fully-specified and refinements in the requirements analysis phase in detail [4]. UCM exhibits the various patterns of behavior that are adequate in capturing the system testing activities. Activities related to system testing can be estimated using UCM.

Researchers have contributed towards estimation of the testing effort using different UML diagrams and specifications that are available at the various phases of software life cycle. Veenendaal [5] proposed the test point analysis based on the functional requirements. Nageswaran [6] applied the use case points to estimate the effort involved in software acceptance testing based on the use case diagram. Baudry *et al*. [7] proposed the testing effort estimation based on the UML class diagram. Aranha *et al*. [8] proposed an estimation model for test execution effort based on the test specification. Xiaochun *et al*. [9] proposed the test execution effort estimation based on use cases, in the context of software testers. Though these approaches estimate the test effort of the software systems at the various phases of the software life cycle, they do not adhere to any standard. On the other hand, FPA is a standard proposed by IFPUG. Hence, it is beneficial to combine the advantages of UCM as well as those of FPA for system test size estimation.

The objective of this work is to propose a system test size estimation method by adapting the FPA method to UCM. This has been achieved by extracting the necessary information from the UCM and deriving the essential components for estimation. The FPA components are appropriately mapped to the derived UCM components and the formulae for test size estimation have been derived. The proposed estimation method has been applied on sample OO projects and the results are empirically evaluated to analyze the accuracy of the estimation.

The remainder of the paper is organized as follows. Section 2 surveys the work related to the estimation of testing effort. In Section 3, the proposed system test size estimation method is presented. Section 4 applies this estimation approach for a case study. Section 5 discusses the empirical results and the analysis of this approach. Section 6 concludes and suggests possible future directions.

2. Related Work

Estimation of testing effort has been proposed by several researchers based on software requirement specification [10], static class diagram [7], UCM [6], [9], [11], [12], functional requirements [13], test specification [8], and complexity [13] [14]. Table 1 outlines the various existing test estimation methods in literature discussed in this section. Veenendaal *et al*. [5] proposed test point analysis based on the functional requirements at the early stage of the software life cycle. Dynamic and static test points were calculated for each identified function of the software system. The estimation corresponding to a dynamic test point for a function includes function point, function dependent, and quality dependent factors. Similarly, the estimation corresponding to a static test point for a function includes sixteen quality checklists in ISO 9126. Final test point comprises of dynamic and static test point. The primary test hours estimation depends on test points, total test hours for test points, and environmental factors. Thus, the rough test estimation for a software system based on functions and quality is presented to the client.

Table 2. Comparison of Existing Test Estimation Methods

Author(s)	Proposed work	Drawbacks
Almeida *et al*.	Proposed a method for test effort, based on use cases. It used the parameters namely actor ranking, technical and environmental factors related to testing like test tools, input, environment, distributed system, interface, etc for the calculation of test effort. It used use case point for estimation.	Only the use case is considered for estimation as a whole for determining the weightage. Fine granularity of each use case was not considered in determining the weightage of a use case.
Aranha *et al*.	Proposed an estimation model for test execution effort based on the size and execution complexity measured from test specification written in a controlled natural language.	Test procedure needs to be written in a controlled natural language.
Ashish *et al*.	Proposed test metrics to compute the complexity of requirements on the basis of software requirement specification.	The knowledge of NLP is required to compute metrics.
Baudry *et al*.	Proposed the testing effort measurement based on UML class diagram. measures the complexity of interactions that must be covered during testing.	This work requires an elaborate design to measure the complexity of the interactions.
Nageswaran	Presented a use case based approach for acceptance test effort estimation in V-model developmental life cycle.	Fine grain information corresponding to the various scenarios in use case diagram is not considered in assigning weights to a use case.
Kushwaha *et al*.	This work has made use of an existing simple cognitive metric that includes all the important parameters of software required for estimation of test effort. Demonstrates that the cyclomatic complexity number increases with the increase in the software complexity when a new component is added to existing software. Adapts the cyclomatic metric for the proposed metric.	Authors claim that the proposed metric is robust, but how the robustness was obtained was not clearly stated.
Veenandaal *et al*.	Proposed the test point analysis based on the functional requirements at the early stage of the software life cycle. A rough test estimation for a software system based on functions and quality is presented to the client.	Validation of this work was not discussed.
Xiaochun *et al*.	Estimated the test execution effort estimation. This approach provided two key parts, namely test case number prediction model and test suite execution vector model.	This estimation presented an experience based approach.
Zhou Bo *et al*.	Proposed test suite size estimation based on the use case. The test suite size was predicted from test case number prediction.	Presented an experience based approach for the test suite size estimation.

Nageswaran [6] had estimated acceptance test effort in V-model developmental life cycle through use case diagram. An existing developmental estimation method of use case point has been applied. In this method, the actors and use cases were categorized and weights were assigned to estimate the unadjusted use case point. Technical and environment factors were incorporated with this to estimate the adjusted use case point. By applying the adjusted use case points, the acceptance test effort for EJB / COM / DCOM software had been computed. Though this method employs the use case diagram and predicts the test effort at the analysis phase, it has certain drawbacks. It lacks the description for classification of weights assigned to the technical and environment factors. Further, the fine grain information corresponding to the various scenarios is not considered in assigning weights to a use case.

Aranha *et al*. [8] proposed an estimation model for test execution effort based on the test specification. A measure of test size and execution complexity was defined and validated. Test procedure for this estimation had been written in controlled natural language. A test suite comprises of many test cases and each test case has been analyzed for functional and non-functional characteristics to obtain the execution points. Eight functional and three non-functional characteristics have been considered in this work. Based on the productivity of the test team, from the total execution points, the test execution effort is calculated. Guidelines and weightage for the different characteristics have been specified. Justification for the weightage assigned to the various characteristics is rather ambiguous.

Xiaochun *et al*. [9] proposed test effort estimation based on test suite execution vector model and test case number prediction model. This approach introduced a metric known as use case verification point to measure the transactions and entity objects in a use case. Test case number, execution complexity, and tester rank are determined to estimate the test execution effort from use cases. The accuracy of the estimation depends on the expert judgment. Guidelines for the test execution complexity calculation need to be precisely stated. Rules for the use case verification point need to be stated. The procedure, for deriving the test cases, entity object, transaction steps, test set suite, special requirements and use case verification point, is not clear.

Ashish *et al*. [15] proposed test metric that computes the requirement based complexity using software requirement specification document. Ashish *et al*. [10] has empirically proposed test metrics for the estimation of software test effort using software requirement specification. McCabe proposed a graph theoretic complexity measure based on the source code. Zhou *et al*. [12] present an experience based approach for the test suites size estimation based on use case. Almeida *et al*. [11] proposed a method to extract test activity effort based on the use case. Aranha *et al*. [8] proposed an estimation model for test execution effort based on test specification. Baudry *et al*. [7] proposed the testing effort estimation based on UML class diagram. Kushwala *et al*. [14] discusses cognitive information complexity measure and modeling of test effort based on Commercial off The Shelf (COTS) components and Component Based Software Engineering (CBSE). Table 2. represents the comparison of various existing test estimation methods. From all these works, it is evident that, the use case model data is not completely utilized in estimating the system test size. Thus, a new estimation technique has been proposed in this paper that integrates the use case diagram and use case specification document by adapting FPA to estimate the system test size of the object oriented software at the analysis phase.

3. System Test Size Estimation

The proposed estimation method focuses on the various features of the UCM such as actors, use cases within the system boundary, external references, relationship between use cases, and transaction features. This section describes the proposed approach for determining the system test size of the object oriented software, based on UCM. The following subsection describes these steps in detail. The steps involved are:

1) Convert the use case diagram into the use case graph.
2) Derive the test set suite from the use case graph.
3) For the possible interactions based on the edges, determine the estimation parameters for STSP components.
4) Map the FPA components to STSP components for estimation
5) Classify the technical complexity factors for system testing.
6) Estimate the system test point.

3.1. Formation of Test Set Suite

In the use case diagram, the system boundary identifies the border between different applications. Fig. 1 gives a pictorial overview of the boundary identification of the developmental software. The boundary is identified for the proper classification of logical files. Different actors, such as an active actor, a passive actor or a device will communicate with the use cases for various operations in the system boundary. These use cases in turn may invoke other services outside the system boundary.

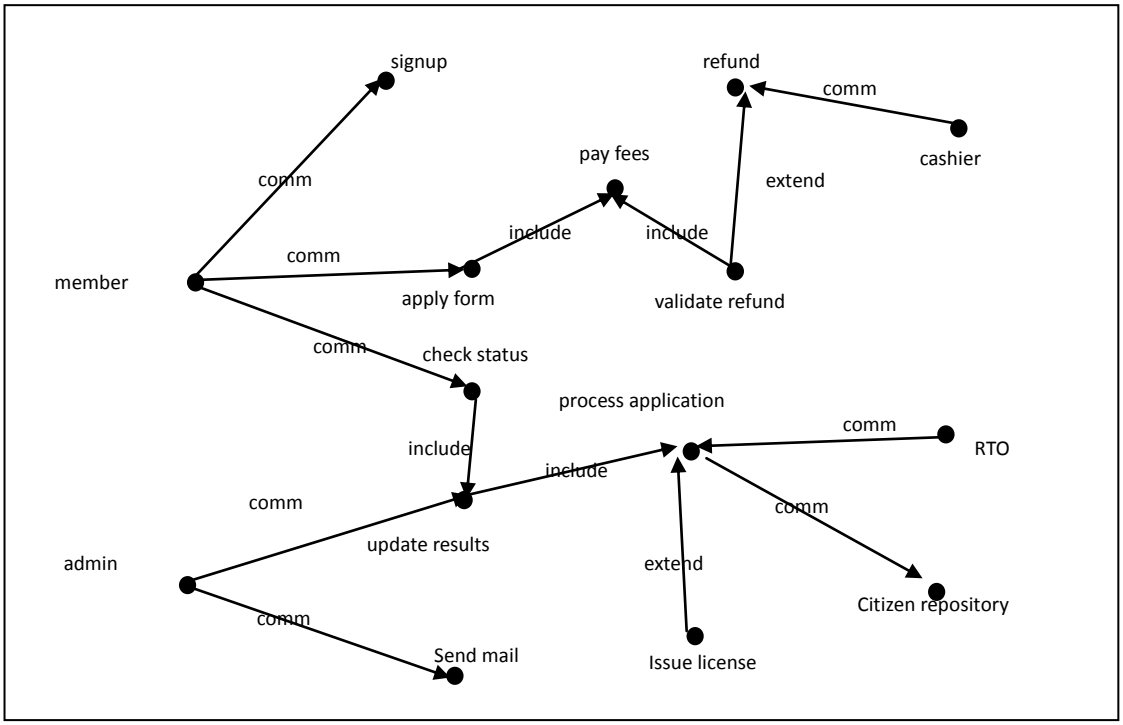

Fig. 1. Use case graph for RTO application.

The case study of Regional Transport Office (RTO) is a software application that manages all the activities in RTO for issue of license. RTO is an on-line system which enables people to apply for learner's permit, license, renewals, vehicle registrations and transfers. It also provides them with the necessary information regarding the dates on which tests will be conducted, issue dates, and expiry time limits. Alert mails will be automatically sent to people whose licenses are about to expire. It allows users to pay registration fees online using their credit/debit cards. The use case diagram for this application is converted into a use case graph, G and it is represented in Fig. 1. Each use case and actor is represented as a vertex, and the relationships such as *'uses'* or *'include', 'extend'*, and *'communicate'* alias *'comm'* are represented as edges in a use case graph.

Definition 1: Use case graph: *Connections between the vertices and edges constitute a use case graph.*

A use case graph has *'n'* vertices and *'m'* edges.

Definition 2. Test Set: *Information exchanged between two vertices in each edge in the use case graph G is a*

Test Set (TS).

Let TS be test set. It has $vertex_i$, $edge_a$, $vertex_j$ ε G, where $1 \le i \le n$, $1 \le j \le n$, $1 \le a \le m$ and for all test sets, $i <> j$, TS = { $vertex_i$, $edge_a$, $vertex_j$ }

Definition 3. Test set suite: The set of all qualified test sets in the connected or disconnected components of the use case graph, is defined as the Test Set Suite(TSS).

TSS has a set of TS_i, where $1 \le i \le n^2$, TSS = {{TS_1},...,{TS_i}}. The qualified test sets for test set suite are populated by the following steps.

1) Test sets for *'extend'* and *'include'* edges are populated in the test set suite.
2) Each vertex in the test set for *'comm'* edge is checked for whether it is already populated as a test set in the test set suite.
3) If this is true, *'comm'* edge test set need not be included, otherwise it should be included in the test set suite.

Here afterwards, the test set means qualified test set.

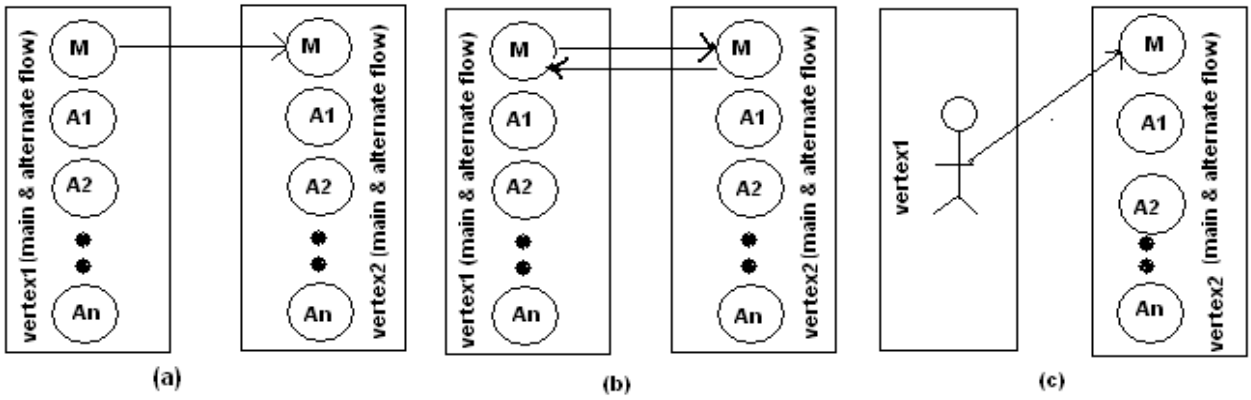

Fig. 2. Interaction for edges (a)*'extend'* (b)*'include'* (c) *'comm'.*

3.2. Interactions of Test Set

UCM has use case specification document that describes all the use cases in the use case diagram. Each use case in the use case specification document has details such as input data, pre-condition, post-condition, main flow, alternate flow, and exception flow.

Definition 4. Main flow: *Each vertex in use case graph G has a well defined goal and this is defined as the main flow.*

Main flow for a $vertex_i$ is expressed as M.

Definition 5. Alternate flow*: If there is any obstacle to achieve the main flow of each vertex in use case graph G, there is deviation and this is defined as alternate flow.*

An alternate flow for a $vertex_i$ is expressed as A.

Definition 6. Execution Sequence*: The interaction of flows based on the type of edge between the vertices in a test set is known as Execution SEQuence (ESEQ).*

Definition 7. Test Step : *A message in the interaction of flows in a test set is Execution Step (ES).*

Definition 8. Data Flow : *Data used in ESEQ in a test set is Data Flow(DF).*

In case of the test set with *'extend'* edge, the completion of execution steps of the main flow of $vertex_i$ will invoke the execution steps of the main flow of $vertex_j$. The interaction between the main flows and alternate flows for this edge is shown in Fig. 2(a).

Based on the main flow and alternate flow, for each test set with this edge, the following three possible interactions of flows are to be considered, to identify the total number of ESEQ and ES.

1) Main flow interactions of vertices.
2) Main flow of $vertex_i$ to one of the many alternate flows in $vertex_i$.
3) Main flow of $vertex_i$ to main flow of $vertex_j$ followed by one of the many alternate flows of $vertex_j$.

In case of the test set with *'include'* edge, the main flow of $vertex_i$ is the caller and main flow of $vertex_j$ is the callee. Any one of the execution steps of the caller will invoke the execution steps of the main flow of the callee. On completion, the control returns back from the callee to the caller. The interaction between the main flows for this edge is shown in Fig. 2(b). Based on the main flow and alternate flow, for each test set with this edge, the following four possible interactions of flows are to be considered, to identify the total number of ESEQ and ES.

4) Main flow interactions of $vertex_i$ to $vertex_j$ and then back to $vertex_i$.
5) Main flow of $vertex_i$ to one of the many alternate flows of $vertex_i$.
6) Main flow of $vertex_i$, main flow of $vertex_j$ followed by one of the many alternate flows of $vertex_j$.
7) Main flow of $vertex_i$, main flow of $vertex_j$ followed by one of the many alternate flows of $vertex_j$ and then one of the many alternate flows of $vertex_i$.

In case of the test set with *'comm'* edge, as one vertex is an actor, the main flow and alternate flows of the other vertex are considered. Interaction between them is pictorially shown in Fig. 2(c). Based on the main flow and alternate flow, the following two possible interactions of flows need to be considered, to identify the total number of ESEQ and ES.

8) Main flow interactions of vertices.
9) Main flow of $vertex_j$ to one of the many alternate flows in $vertex_j$

Table 3. STTCF in System Testing

Terms	Factors	Brief Description	Wt
FF	Test environment	How many setups need to be undertaken for establishing the environment?	5
	Test documentation	How many associated documents need to be prepared?	5
	Test conditions	What are the test scenarios to be verified for the completion of transactions?	5
	Test ware	What storage mechanism is adapted to store the different artifacts of testing?	5
NFF	Performance	What are the different test scenarios for which the response rate needs to be measured?	5
	Load testing	What are the different types of scenarios for which response time has to be measured?	5
	Security testing	How many unauthorized accesses need to be prevented?	5
	Stress testing	What different evaluation mechanisms are used to study the system behaviour?	5
OF	Testing tool	How many tools are used for testing?	5
	Tester's skill	How many unskilled testers are involved in testing?	5
	Innovative technology	What are the hurdles in setting up innovative technology?	5
	Multiple test configurations	What are the test configurations used for testing?	5
	Geographical distribution of team	How many continents are used?	5
	Product size	What is developmental size of the software to be tested?	5

3.3. System Test Size Point Estimation

Function points introduced by Albrecht *et al*. [1] are used in software industry to estimate the functionality of the software. In 1984, Albrecht refined the method and several versions of the function point counting practices manual have been published by the International Function Point User Group (IFPUG). It consists of five components namely Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO), External Inquiry (EQ), and fourteen Technical Complexity Factors (TCF). The FPA components of ILF and EIF are mapped to Internal Relational File (IRF) and External Relational File (ERF) respectively, while EI component is mapped to Data Transaction (DT) of System Test Size Point (STSP) component. EO and EQ components of FPA have no direct relationship in this context, and are not considered. The STSP components are discussed in detail below.

Definition 9. Internal Relational File: *Information exchanged in a test set, internally within the boundary of use case graph G is defined as an Internal Relational File (IRF).*

Thus, a test set is identified as one IRF. Classification of an IRF depends on two parameters namely ESEQ and ES. The weightage for each ESEQ and ES is one. Based on the number of ESEQ and ES, the complexity of IRF, is classified as low, average or high by applying the IFPUG, ILF complexity table. Summation of the complexity of each IRF yields the total complexity due to IRF. IRF complexity estimation for test sets based on different categories of edges for the case study is discussed in Section 4.

Definition 10. External Relational File*: Information exchanged in a test set, externally away from the boundary of use case graph G is defined as an External Relational File (ERF).*

In case, if any vertex in the use case graph G, references another vertex externally, a separate test set is created. Each such identified test set is one ERF. Classification of an ERF depends on two parameters namely ESEQ and ES. The weightage for each ESEQ and ES is one. The complexity of each ERF, based on the weightage assigned to ESEQ and ES, is classified as low, average or high by applying the IFPUG, ELF complexity table. Summation of the complexity of all ERF yields the total complexity due to ERF.

Table 4. Test Set Suite for RTO Application

No.	Test Set	No.	Test set
1	{apply form, include, signup}	6	{update result, include, process application}
2	{apply form, include, pay fees}	7	{process application, extend, issue license}
3	{pay fees, include, validate refund}	8	{applicant, comm, signup}
4	{validate refund, extend, refund}	9	{admin, comm, send mail}
5	{check status, include, update results}	10	{process application, comm, citizen repository}

Definition 11. Data Transaction: *Data exchange in a test set is DT.*

Each identified test set in the use case graph, exchanges data between the vertices. Classification of a DT depends on two parameters namely ESEQ and DF. The weightage for each ESEQ and DF is one. Based on the number of ESEQ and DF, the complexity of each DT, is classified as low, average or high by applying the IFPUG, EI complexity table. Summation of the complexity of each DT yields the total complexity due to DT.

Definition 12. System Testing Technical Complexity Factors: *The parameters that affect the system testing that can be quantified are known as System Testing Technical Complexity Factors (STTCF).*

STTCF identified for system test size estimation are tabulated in Table 3. They are broadly classified into three categories such as Functional Factors (FF), Non-Functional Factors (NFF), and Organizational Factors (OF). The assignment of weightage to these factors depends on their impact on testing. The weightage for these factors varies in the range of 0 to 5 corresponding to no influence, incidental, moderate influence, average influence, significant influence, and strong influence respectively. There are four factors namely test environment, test documentation, test conditions and testware that are dependent on functional

characteristics of the testable software. There are also four factors namely performance, load, security, and stress that are dependent on non-functional characteristics of the testable software. There are totally six factors namely testing tool, tester's skill, innovative technology, multiple test configurations, geographical distribution of team, and product size that are dependent on organizational decisions for testing. All these factors and their contribution towards testing are summarized in Table 3. The STSP estimation of an object oriented software application is calculated as shown below.

$$\text{STSP} = [a\times (\text{IRF} + \text{ERF} + \text{DT})] \tag{1}$$

where

$$a = 0.65 + \left(0.01 \times \sum_{i=1}^{14} STTCF_i\right)$$

$$\text{IRF} = f(\text{ESEQ}, \text{ES}),\ \text{ERF} = f(\text{ESEQ}, \text{ES}),\ \text{DT} = f(\text{ESEQ}, \text{DF})$$

STSP is determined from four components namely ERF, IRF, DT and STTCF. Next section describes how the proposed size estimation technique has been applied to the chosen RTO application case study.

Table 5. Use Case Documentation for a Test Set with *'Extend'* Edge

Use case name:	Process application	Issue license
Main Flow:	M_3. Issue the LLR for the user M_3E_1 Validate the application form, in case of invalid go to step A31E1 M_3E_2 Confirm the payment details, in case of incorrect payment go to step $A_{32}E_1$ M_3E_3 Confirm the test dates M_3E_4 Issue the LLR for new user M_3E_5 Update the user status	M_4. Issue the License for the user M_4E_1 Generate the license for the user, in case of failure go to step $A_{41}E_1$ M_4E_2 Update the RTO information M_4E_3 Update the user status
Alternate flow:	A_{31} Invalid application form $A_{31}E_1$ Convey the user to submit a valid application form A_{32} Incorrect payment $A_{32}E_1$ Convey the user to pay the correct amount	A_{41} License not issued $A_{41}E_1$ License not issued for the user

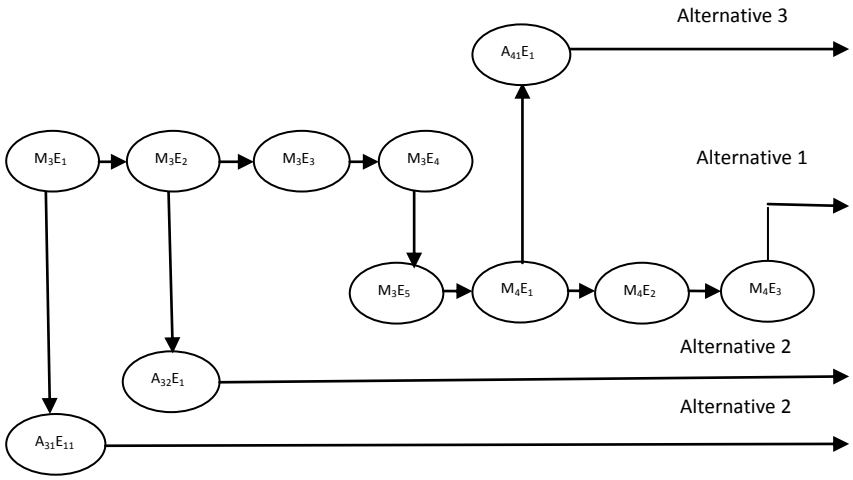

Fig. 3. Interactions for *'extend'* edge.

4. Estimation Validation

Consider the use case graph, G shown in Fig. 1 for RTO application developed in software engineering laboratory using object oriented methodology. This use case graph, has vertices such as *applicant, admin, cashier, RTO, Citizen repository, apply form, signup, check status, update results, send mail, pay fees, refund, validate refund, process application, issue license and edges such as include, extend, comm*. Based on the definition 2, the identified initial test sets from the use case graph are as follows: *{applicant, comm, signup},{applicant, comm, apply form},{applicant, comm, check status}, {apply form, include, signup}, {apply form, include, pay fees}, {payfees, include, validate refund}, {validate refund, extend, refund}, {refund, comm, cashier}, {check status, include, update results}, {admin, comm, send mail},{update result, include, process application}, {process application, extend, issue license}, {process application, comm, RTO}, {process application, comm, Citizen repository}* . The test set suite contains the qualified test sets based on the definition 3 and it is represented in Table 4. Main flow and alternate flow for *'extend'* and *'include'*, edges are provided in the Table 5 and 6 respectively.

4.1. Internal Relational File Complexity Estimation

The estimation of IRF for a test set in Table 4, *{process application, extend, issue license}* with *'extend'* edge is as follows. Using the Table 5, it is clear that *'process application'* vertex comprises one main flow (M_3) and two alternate flows (A_{31}, A_{32}), while *'issue license'* vertex comprises one main flow (M_4) and one alternate flow (A_{41}). M_3 contains five execution steps E_1...E_5, while M_4 contains three execution steps E_1...E_3. Alternate flows A_{31}, A_{32}, A_{41} each contain a single execution step E_1. The three interactions discussed in Section 3.2 for the *'extend'* edge are applied to identify the total number of ESEQ and ES parameters for estimating the IRF for this test set.

Table 6. Use Case Documentation for a Test Set with *'Include'* Edge

Use case name:	Apply form	Pay fees
Main Flow:	M_1 User submits the new license form M_1E_1 Applicant request new license form M_1E_2 Display the new license form M_1E_3 Complete and submit the license form. M_1E_4 Check whether license has been issued already. If already issued go to step $A_{11}E_1$. M_1E_5 Check the status of license. If status already exists go to step $A_{12}E_1$ otherwise modify it. M_1E_6 Generate a new application number. M_1E_7 An applicant receives a new application number for the new application form submitted. M_1E_8 An applicant initiates the payment process to pay the fees for the license. M_1E_9 Go to step M_2E_1 to process the payment. M_1E_{10} License form submitted if payment complete, else go to step $A_{13}E_1$	M_2 Complete payment process M_2E_1 Applicant chooses the gateway M_2E_2 Payment gateway is checked M_2E_3 Payment form is displayed M_2E_4 Applicant fills the card and amount information. M_2E_5 Submits the payment form. M_2E_6 On payment success the applicant receives receipt and control transfers to M_1E_{10} else to $A_{21}E_1$
Alternate flow:	A_{11} User already a license holder $A_{11}E_1$ Convey the applicant that he is already a license holder. A_{12} User has already applied for license. $A_{12}E_1$ Convey the applicant that he is already applied for license. A_{13} Incomplete submission $A_{13}E_1$ Convey the applicant incomplete license form submission.	A_{21} Amount not deducted $A_{21}E_1$ Convey payment failure to the applicant and go to step M_1E_{10} A_{22} Submits an incomplete form $A_{22}E_1$: The applicant submits the incomplete form for the payment process

For the first interaction, the execution steps of M_3 are executed followed by the execution steps of M_4. The total number of ESEQ and ES identified for this interaction are 1 and 8 respectively. The corresponding

ESEQ is M_3E_1...M_3E_5, M_4E_1...M_4E_3. For the second interaction, when the execution steps of M_3 are executed, there is a possibility that any one of the alternate flows, either A_{31}, or A_{32} can be invoked. The total number of ESEQ and ES identified for this interaction are 2 and 4 respectively. The corresponding ESEQ can be M_3E_1, $A_{31}E_1$ or M_3E_2, $A_{32}E_1$. For the third interaction, the execution steps of M_3 are executed completely followed by M_4. Here, there is a possibility that the alternate flow A_{41} can be invoked. The total number of ESEQ and ES identified for this interaction are 1 and 7 respectively. The corresponding ESEQ is M_3E_1...M_3E_5, M_4E_1, $A_{41}E_1$. The complete ESEQ for the three interactions of this test set is represented in Fig. 3. Thus, the total number of ESEQ and ES identified for this test set are 4 and 19 respectively. Based on these values, the complexity for IRF is 7 using the ILF table defined in IFPUG.

The estimation of IRF for a test set *{apply form, include, pay fees}* with *'include'* edge is as follows. The use case documentation for this test set is captured in Table 6. From the Table 6, it is clear that *'apply form'* vertex comprises one main flow (M_1) and three alternate flows (A_{11}, A_{12}, A_{13}), while *'pay fees'* vertex comprises one main flow (M_2) and two alternate flows (A_{21}, A_{22}). M_1 contains ten execution steps E_1...E_{10}, and M_2 contains six execution steps E_1...E_6. Each alternate flow A_{11}, A_{12}, A_{21}, and A_{22} contains a single execution step E_1. The four interactions identified for this *'edge'* in Section 3.2 are applied to calculate the total number of ESEQ and ES parameters for estimating IRF corresponding to this test set.

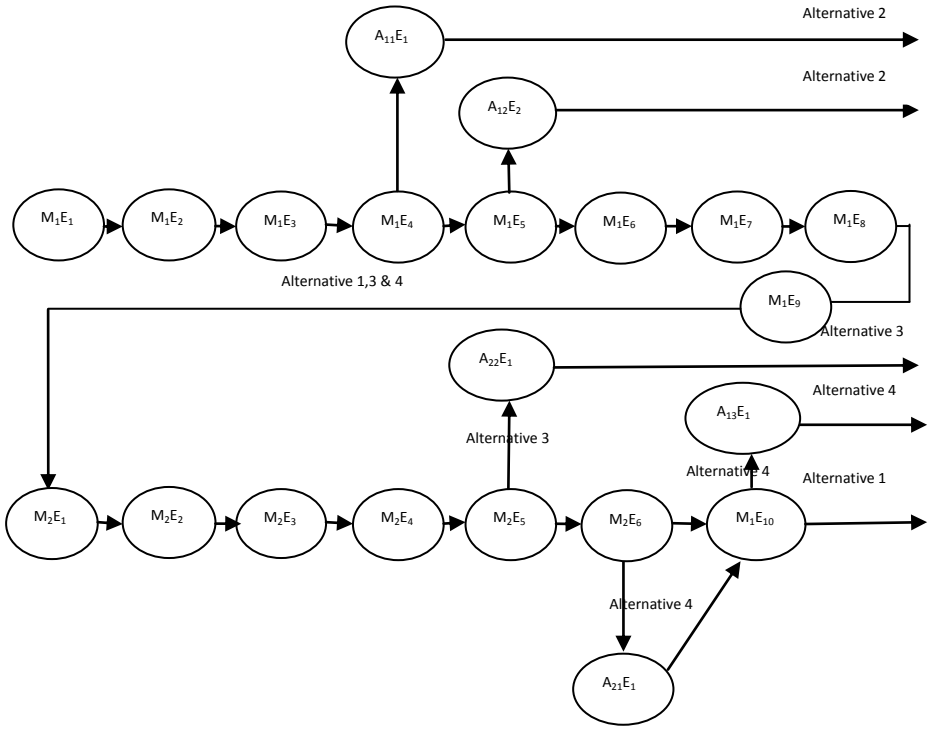

Fig. 4. Interactions for *'include'* edge.

For the first interaction, the execution step $vertex_i$ (caller) will invoke $vertex_j$ (callee) execution step. The E_1...E_9 steps of M_1 followed by E_1...E_6 of M_2 are executed and the control transfers back to the caller, M_1E_{10}. Number of ESEQ and ES identified for this interaction are 1 and 16 respectively. The corresponding ESEQ is M_1E_1...M_1E_9, M_2E_1...M_2E_6, M_1E_{10}. For the second interaction, there is a possibility that any one of the alternate flows of $vertex_i$ (caller) can be invoked from main flow of $vertex_i$(callee). Number of ESEQ and ES identified for this interaction are 2 and 11. The corresponding ESEQ can be M_1E_1...M_1E_4, $A_{11}E_1$, or M_1E_1...M_1E_5, $A_{12}E_2$. For the third interaction, any one of the execution steps of $vertex_i$ (caller) main flow will invoke an execution step of $vertex_j$(callee). From this, there is a possibility that any one of the many

alternate flows of vertex$_j$ will be invoked. Number of ESEQ and ES are 1 and 16 respectively. The corresponding ESEQ is M_1E_1...M_1E_9, M_2E_1...M_2E_5, $A_{22}E_1$.

For the fourth interaction, any one of the execution steps of *vertex$_i$* (caller) main flow will invoke an execution step of *vertex$_j$*(callee). Here, there is a possibility that any one of the many alternate flows of *vertex$_i$* and *vertex$_j$* will be invoked. Number of ESEQ and ES are 1 and 18 respectively. The corresponding ESEQ is M_1E_1...M_1E_9, M_2E_1...M_2E_6, $A_{21}E_1$, M_1E_{10}, $A_{13}E_1$.Thus, the total number of ESEQ and ES, identified for this test set are 5 and 61 respectively. The complete execution sequence for the four interactions of this test set is represented in Fig. 4. With the identified values of ESEQ and ES, the complexity for IRF is 10 using the ILF table defined in IFPUG.

4.2. External Relational File Complexity Estimation

In RTO application, vertex *'issue license'* references another vertex *'Citizen repository'* externally, which is defined in another application. The qualified test set identified for ERF is *{process application, comm, Citizen repository}.* The estimation of ERF for this test set with *'comm'* edge is as follows. The vertex *'process application'* documentation is represented in Table 5. From Table 5, it is clear that *'process application'* vertex comprises one main flow (M_3) and two alternate flows (A_{31},A_{32}), while *'Citizen repository'* vertex is externally referenced and is a part of another application. The number of execution steps in the main flow M_3, is three and the alternate flows A_{31} and A_{32} is one each. Based on the discussion in Section 3.2, for *'comm'* edge test set, the two possible interactions need to be considered, to classify the total number of ESEQ and ES. For the first interaction, the number of ESEQ and ES are 1 and 5 respectively. The corresponding ESEQ is M_3E_1, ...M_3E_5. For the second interaction, the number of ESEQ and ES are 2 and 5 respectively. The corresponding ESEQ is M_3E_1, $A_{31}E_1$ or M_3E_1, M_3E_2, $A_{32}E_1$. Thus, the total number of ESEQ and ES estimated for this test set are 3 and 10 respectively.

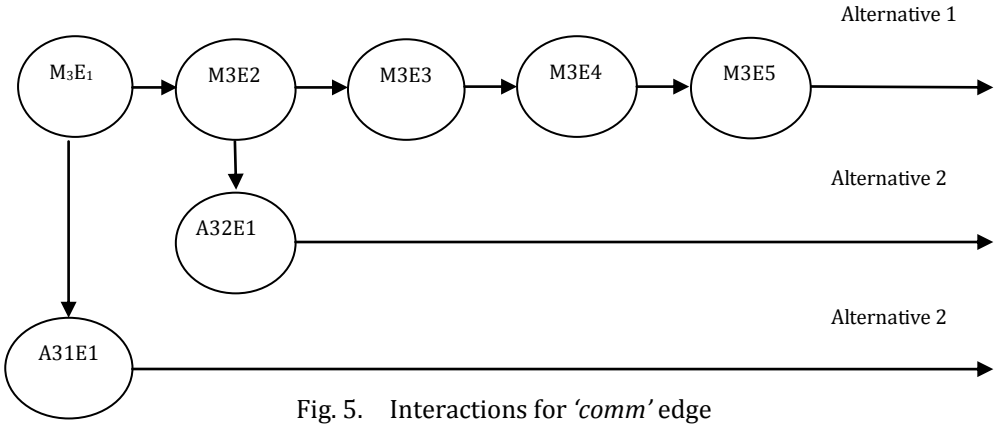

Fig. 5. Interactions for *'comm'* edge

The complete execution sequence for the two interactions of this test set is represented in Fig. 5. With the identified values of ESEQ and ES, the complexity for ERF is 7 using the EIF table defined in IFPUG.

4.3. Data Transaction Complexity Estimation

The estimation of DT for a test set *{applicant, comm, sign up}* is as follows. ESEQ is identified for this test set as 2. The identified DF for this test set is login name, login password, question, hint, email-id. Cumulatively, adding these yields a DF value of 5. With the identified values of ESEQ and DF, the complexity for DT is 4 using the EI table defined in IFPUG. The complete IRF, ERF, DT, and STTCF for the RTO application are 101, 7, 53 and 13 respectively and these values are shown in Table 7. Applying these values in eqn(1), the STSP is estimated as 125.58 TSP. For the same software, UCMFP[1] was estimated at 87.74

FP.

5. Empirical Results and Analysis

5.1. Test Data Analysis

Table 7. STSP Estimation Parameters Values for RTO Application

Test no	Functional Parameter Estimation							System Testing Technical Complexity Factors Estimation		
	File	Files			DT			Term	Factor	Weight
		ESEQ	ES	Complexity	ESEQ	DF	Complexity	FF	Test environment	1
1	IRF	6	38	15	6	23	6		Test Documentation	2
2	IRF	5	61	5	5	21	6		Test Conditions	3
3	IRF	5	26	10	5	11	6		Testware	1
4	IRF	5	26	10	5	12	6	NFR	Performance	2
5	IRF	7	29	15	7	22	6		Load Testing	2
6	IRF	6	24	15	6	32	6		Security Testing	2
7	IRF	4	19	7	4	29	6		Stress Testing	0
8	IRF	2	7	7	2	5	4	OF	Testing Tool	0
9	IRF	2	8	7	2	3	3		Testers Skill	0
10	IRF	3	10	7	2	8	4		Innovative Technology	0
Total Complexity				108			53		Multiple Test Configurations	0
									Geographical Distribution of Team	0
									Product Size	0
								Total Complexity		13

Table 8. Test Size Estimation

Project	Number of use cases	Complexity due to		STSP	UCMFP	UCMFP Development Effort (hrs)	STSP Test Effort (hrs)
		IRT & ERT	DT				
P1	13	75	46	94.91	83.46	2.1162	15.4229
P2	13	86	54	109.82	118.77	38.5884	17.8458
P3	16	91	66	122.59	116.63	37.8931	19.9209
P4	13	79	51	101.72	110.21	35.8072	16.5295
P5	11	62	36	76.26	82.39	26.7685	12.3923
P6	10	69	49	92.53	85.6	27.8114	15.0361
P7	11	84	52	106.08	98.44	31.9832	17.238
P8	11	83	61	112.04	107	34.7643	18.2065
P9	12	77	49	98.14	104.86	34.0690	15.9478
P10	10	84	42	88.23	83.46	27.1162	14.3374
P11	14	68	46	104.9	113.42	36.8502	17.0463
P12	13	82	52	109.45	104.86	34.0690	17.7856
P13	12	64	43	83.34	90.95	29.5497	13.5428
P14	10	72	49	94.56	87.74	28.5067	15.366
P15	12	79	63	111.65	105.93	34.4167	18.1431

The proposed estimation approach was applied to few of the OO software projects developed in our software engineering laboratory. These projects were chosen for analysis because all of them were developed in the same environment, by following all the activities in software life cycle. Each team followed the same procedure to collect the data required for empirical analysis. All these projects were developed by the final year undergraduate students using OO languages. Table 8 shows details of the projects chosen for training. This data is applied in equation (1) and STSP in terms of TSP is estimated and tabulated in

Table 8. Developmental size for these projects was also estimated by applying the UCMFP [1] in terms of FPs and the results are tabulated in Table 8. Since the FPA is adapted to predict the STSP, the software testing convention factor, followed in our software engineering laboratory, 1 FP=0.1625 man-hours is applied to estimate the system test effort.

Similarly, the software development convention factor, 1FP=0.3249 man-hours is applied to estimate the development effort. Appendix A details the calculation of these convention factors. Table 8 represents the STSP and UCMFP obtained for the fifteen projects and their corresponding development and testing effort.

Pearson correlation coefficient [16] measures the strength of a linear relationship between two projects. The formula used to compute Pearson coefficient is as follows.

$$r = \Sigma\ (Xi - \overline{X}\ )\ (\ Yi - \overline{Y}\ )\ /\ N\ \sigma_x\ \sigma_y$$

In which, '*X*' denotes the STSP test effort, '*Y*' denotes the UCMFP development effort, '*N*' denotes the total number of projects, '*бx*' is the standard deviation of STSP test effort, and '*бy*' is the standard deviation OF UCMFP development effort. The correlation coefficient is always between -1 and +1. The relationship between STSP test effort and UCMFP development effort was analyzed for the fifteen training projects and its value is 0.827 positive. This indicates that a strong positive association exists between them.

Table 9. Effort Prediction for Sample Projects

Project	UCMFP development effort	STSP Actual Test Effort (hrs)	STSP Predicted Test Effort (hrs)	Absolute Error
P1	27.1162	15.4229	13.9904	1.4325
P2	38.5884	17.8458	19.0381	1.1923
P3	37.8931	19.9209	18.1819	1.739
P4	35.8072	16.5295	17.8274	1.2979
P5	26.7685	12.3923	14.5287	2.1364
P6	27.8114	15.0361	14.4324	0.6037
P7	31.9832	17.238	16.1053	1.1327
P8	34.7643	18.2065	17.1803	1.0262
P9	34.0690	15.9478	17.0908	1.143
P10	27.1162	14.3374	14.2241	0.1133
P11	36.8502	17.0463	18.2655	1.2192
P12	34.0690	17.7856	16.9288	0.8568
P13	29.5497	13.5428	15.3935	1.8507
P14	28.5067	15.366	14.7142	0.6518
P15	34.4167	18.1431	17.0498	1.0933
Mean				1.2629

5.2. Test Data Evaluation

Leave one out cross validation technique [16] has been used to evaluate the model. Here, *n*-1 projects are used for training and the remaining one was used for testing. This is repeated for each sample in the sample set and the testing was done using regression technique. Linear regression [16] is a statistical analysis technique used to assess the association between the STSP test effort and UCMFP development effort data that is given in Table 8. The linear regression equation used here is $y = a + bx$. Keeping UCMFP development effort as the independent variable and STSP test effort as the dependent variable, the linear regression equation has been derived from a set of fourteen training projects. The derived equation is tested with the fifteenth project, *a*(derived constant), *b*(derived constant), are applied and predicted STSP test effort is obtained. Likewise, each time, a different set of fourteen projects are used for training and the remaining project is used for testing. The tested sample set of the projects with the corresponding predicted STSP test

effort results are tabulated in Table 9. The mean absolute error measures the accuracy of the tested data for the sample set. The error in tested estimation ranges from 0.1133 to 2.1364 and the mean error found to be 1.2629. Hypothesis testing is used to evaluate experimental outcome about the relationship between STSP actual and STSP predicted test effort data. Null hypothesis (Ho) and the alternative hypothesis (Ha) concerning to this experiment are stated as follows:

Ho: D = 0 No difference exists between the actual and predicted test effort.

Ha: D ≠ 0 Difference exists between the actual and predicted test effort.

The hypothesis is applied on the tested projects using the paired t-test. Paired t-test is a statistical technique that is used to compare two samples that are correlated. It can be used if the sample sizes are very small as long as the pairs are not reliably different.

In this context, the paired t-test is applied to compare between the actual and predicted test effort. The formula used for the paired t-test with n-1 degrees of freedom is:

$$t = \frac{\overline{d}\sqrt{n}}{S}$$

where $\overline{d}$ is the mean difference between two samples, S is the standard deviation of differences, and n is the sample size.

The level of significance considered is 95 percent likelihood (α = .05) that a Type I error is not made. Trade-off for choosing a higher level of significance is that it will take much stronger statistical evidence to ever reject the null hypothesis. The outcome of null hypothesis is reject if the computed value of paired t-test is greater than or equal to +2.015 or less than or equal to −2.015 else it is accept. The collected data, actual and predicted STSP test effort of tested fifteen projects are applied on paired t-test formula. The value obtained is 0.9672. Since the t value obtained from paired t-test formula is less than the +2.015 value, the null hypothesis is accepted. There is no significant difference between the actual and predicted STSP test effort. Thus, the effort can be predicted at the early analysis phase if the use case model is complete.

6. Conclusions and Future Work

This paper proposed a new system test size estimation method at the analysis phase namely System Test Size Points for object oriented software. This method is based on the use case model during the analysis phase. Further, it adapts the IFPUG standard, FPA to the use case model for estimation, in contrast to the existing works, which follow different complexity factors to determine the weightage. The use case diagram is converted into a use case graph, and based on the three different types of edges in the graph, the alternatives are derived to identify the relevant components for estimation from the use case model. After applying these components, complexity for these estimation components was derived by assigning weightages to the various parameters following the appropriate FPA complexity tables. Both STSP and the UCMFP were evaluated for the fifteen projects developed by undergraduate students in our department software engineering laboratory. Leave one out cross validation technique was used for training. The relationship between the STSP test effort and UCMFP development effort was derived through linear regression method. The derived linear regression equation was tested on the remaining project to evaluate the accuracy of the system test size estimation. Mean error rate on the tested projects was found to be 1.2629. Hypothesis testing proved that no significant difference exists between the actual and predicted test effort. The limitation of this approach is that currently it has been tested only with a few projects. The validation can be further strengthened by testing with more number of projects. This work also can be extended to measure the testability at the analysis phase of software life cycle.

Appendix

Appendix A. Calculation Of Convention Factors

Project #	# of Use cases	UCMFP(FP)	Actual development effort(hrs)
P1	13	83.46	30
P2	13	118.77	36
P3	16	116.63	36
P4	1	110.21	36
P5	11	82.39	33
P6	10	87.74	27
P7	11	98.44	33
P8	11	107	33
P9	12	104.86	30
P10	10	83.46	33
P11	14	113.42	36
P12	13	104.86	30
P13	12	90.95	27
P14	10	87.74	30
P15	12	105.93	36
Mean		99.724	32.4

99.724 FP = 32.4 man-hours

1 FP = 32.4 man-hours / 99.724

= 0.3249 man-hours

The developmental convention factor, 1 FP = 0.3249 man-hours is considered for developmental effort calculation. According to the thumb-rule used in the industry, the system testing convention factor is half of the development convention factor. Based on this, the system testing convention factor considered is 1 FP = 0.3249/2 = 0.1625 man-hours.

References

[1] Chamundeswari A., & Babu, C., (2013). Developmental size estimation for object oriented software based on analysis model. *International Journal of Software Engineering and Knowledge Engineering*, *23(3)*, 289-308.

[2] Albrecht, A., (1979). Measuring application development productivity. *Proceedings of the IBM Application Development Symposium* (pp. 83-92).

[3] Kusumoto, S., Matukawa, F., Inoue, K., Hanabusa, S., & Maegawa, Y. (2004). Estimating effort by use case points:method, tool and case study. *Proceedings of the Sixth International Symposium on Software Metrics* (pp. 292–299).

[4] Cockburn, A., (2001). *Writing effective use cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

[5] Veenendaal, E. V., & Dekkers, T. (1999). *Test point Analysis: A Method for Test Estimation*.

[6] Nageswaran, S. (2001). Test effort estimation using use case points. *Quality Week*, 1–6.

[7] Baudry, B., & Traon., Y. L. (2005). Measuring design testability of a UML class diagram. *Journal of Information and Software Technology*, 859–879.

[8] Aranha, E., & Borba., P. (2007). An estimation model for test execution effort. *Proceedings of the Proceedings of the First International Symposium on Empirical Software Engineering and Measurement* (pp. 107–116).

[9] Xiaochun, Z., Bo, Z., Fan, W., Yi, Q., & Lu, C., (2008). Estimate test execution effort at an early stage: An empirical study. *Proceedings of the IEEE International Conference on Cyberworlds* (pp. 195–200).

[10] Ashish, S., & Dharmender, S. K., (2012). Applying requirement based complexity for the estimation of software development and testing effort. ACM SIGSOFT Software Engineering Notes, 37(1), 393-415.

[11] Almeida, E. R. C., Abreu, B. T. D., & Moraes, R., (2009). An alternative approach to test effort estimation based on use cases. *Proceedings of the International Conference on Software Testing Verification and Validation* (pp. 279 – 288).

[12] Zhou, W., & Liu, Q. (2010). Extended class point approach of size estimation for OO product. *Proceedings of the International Conference on Computer Engineering and Technology* (pp. 117-122).
[13] McCabe, T. J., (1996). A complexity measure. *Proceedings of the IEEE Transactions on Software Engineering*, *2(4)*, 308-320.
[14] Kushwaha D. S., & Misra, A. K. (2008). Software test effort estimation. *ACM SIGSOFT Software Engineering Notes*, *33(3)*.
[15] Ashish, S., & Dharmender, S. K. (2011). A metric suite for early estimation of software testing effort using requirement engineering document and its validation. *Proceedings of the IEEE Second International Conference on Computer and Communication Technology* (pp. 373-378).
[16] Anderson, R. D., Sweeney, D. J., & Williams, T. A. (2004). *Statistics for Business and Economics* (9th ed.), Mason, OH: South-Western College Publishing.

Chamundeswari Arumugam received her B.E. degree in computer science and engineering from Dr. M.G.R. Engineering College, affiliated to Madras University, Chennai, India in 1992. She received her M.E. degree in software engineering from College of Engineering, Anna University, Chennai, India, in 2002. She completed her Ph.D. degree in in the area of software estimation from Anna University, Chennai, India, in 2013.

She has published many papers in international conference and journals. Her current research interests include software estimation, software testing, and cloud computing, etc. Now she is a professor at Sri Siva Subramaniya College of Engineering, India. She is a life member of Computer Society of India (CSI) and Indian Society for Technical Education (ISTE).

Chitra Babu is a professor and the head of the Department of Computer Science, SSN College of Engineering, Chennai. She received her PhD in computer science from IIT, Madras, Chennai and M.S. in CIS from the Ohio State University, USA. Her research interests include software engineering, service oriented architecture and cloud computing. She has graduated two PhD scholars and is currently guiding four PhD research scholars.

She has published several research papers in international journals as well as conferences. She has served as a technical program committee member in various international conferences. She is a member of ACM, IEEE, CSI and ISTE.