TESTING EFFECTIVENESS AND EFFORT IN SOFTWARE PRODUCT LINES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MERT BURKAY ÇÖTELİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2013

Approval of the thesis:

TESTING EFFECTIVENESS AND EFFORT IN SOFTWARE PRODUCT LINES

Submitted by **MERT BURKAY ÇÖTELİ** in partial fulfillment of the requirement for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Semih Bilgen _____
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt Bazlamaçcı _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Şenan Ece Güran Schmidt _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Ali Hikmet Doğru _____
Computer Engineering Dept., METU

Cumhur Ünlü (M.Sc.) _____
Test Engineering Dept., ASELSAN A.Ş.

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Mert Burkay Çöteli

Signature :

ABSTRACT

TESTING EFFECTIVENESS AND EFFORT IN SOFTWARE PRODUCT LINES

Çöteli, Mert Burkay
M. Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Semih Bilgen

January 2013, 60 pages

Software product lines (SPL) aim to decrease the total software development cost by the help of reusability and variability. However, the increasing number of variations for the delivery types of products would result in increasing cost of the verification and validation process. Total testing cost of development can also be decreased by reusing test cases and scripts. The main objective of this study is to increase testing effectiveness while minimizing testing effort. Four different cases consisting of Aselsan's SPL projects have been studied. Firstly, FIG Basis path method was applied at the functional testing phase, and an increase on the testing effectiveness value has been observed. FIG basis path method is a test case sequence generation technique using the feature tree of the software component. This method would be preferable to improve testing effectiveness on the functional verification phase. The second study was on testing effort estimation. There are two testing approaches for SPL projects, namely infrastructure based and product focused testing. These two techniques have been compared in terms of testing effort. It was a study that gives an idea to test managers about the selection of the proper testing technique. Thirdly, reusability techniques were evaluated. Reusability of testing artifacts can be used to decrease the total testing effort. Two reusability techniques for testing artifacts were compared in terms of the number of test cases. Proper technique would be chosen to decrease testing effort. Finally, selection of a reference application on platform tests was proposed and software products were grouped according to the redundancy values. Then, testing effectiveness values were evaluated for each test grouping.

Keywords: SPL Testing, Testing Effectiveness, Testing Effort, Testing Techniques.

ÖZ

YAZILIM ÜRÜN HATTINDA ETKİLİ TEST VE TEST ÇABASI

Çöteli, Mert Burkay
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Semih Bilgen

Ocak 2013, 60 Sayfa

Yazılım ürün hattı toplam geliştirme maliyetinin yeniden kullanılabilirlik ve değişkenlik yardımıyla düşürülmesi için önerilmiş bir yazılım geliştirme yöntemidir. Fakat teslim edilme çeşitlerine göre değişkenlik gösteren yazılımlardaki artan varyasyonlar yazılımın doğrulanma ve onaylanma aşamasında maliyetin artmasına sebep olmaktadır. Geliştirme aşamasındaki testlerin maliyeti test tanımlarının ve yazılımlarının yeniden kullanılmasıyla düşürülebilir. Bu çalışmanın esas amacı testlerin etkili yapılabilme oranının artmasını test maliyetinin minimize edilerek sağlanmasıdır. Aselsan'ın yazılım ürün hattı projelerini kapsayan 4 durum bu çalışmada işlenmiştir. İlk olarak, FIG basis path metodu yazılım ürün hattı projelerinin fonksiyonel testlerinde uygulanmış ve sonrasında etkili test sonuçlarında artış gözlenmiştir. FIG basis path metodu yazılım bileşeninin yetenek ağacını kullanılarak test tanım sırası oluşturma yöntemidir. Bu metod fonksiyonel testlerin etkili test değerlerini arttırmak için tercih edilebilir. İkinci çalışma test çabası tahminini işleyen bir çalışmadır. Yazılım ürün hattı projelerinde altyapı tabanlı ve ürün odaklı olmak üzere iki farklı test yöntemi uygulanmaktadır. Bu yöntemler test çabası kapsamında bu çalışmada karşılaştırılmıştır. Bu çalışma test yöneticisine uygun tekniğin seçilmesi konusunda fikir verebilir. Üçüncü olarak yeniden kullanılabilirlik teknikleri incelenmiştir. Test tanımlarının yeniden kullanılabilirliği test çabasını düşürmek için kullanılabilir. İki yeniden kullanılabilirlik yöntemi toplam test tanımları kapsamında karşılaştırılmıştır. Uygun yöntem test çabasını düşürmek için seçilebilir. Son olarak platform testlerinde referans bir uygulama seçimi önerilmiş ve ürünler benzerlik değerlerine göre gruplanmıştır. Sonrasında, etkili test değerleri her bir grup için değerlendirilmiştir.

Anahtar Kelimeler: Yazılım Ürün Hattı Testleri, Etkili Test, Test Çabası, Test Teknikleri

To My Parents and To My Brother

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Prof. Dr. Semih Bilgen for his understanding, patience and supervision throughout this thesis. This thesis would not have been completed without his realistic, encouraging and constructive guidance.

I would like to express my appreciation to ASELSAN for providing tools and other facilities throughout this study.

I would like to thank to all my friends and colleagues for their understanding and continuous support during my thesis. Special thanks to TADES Software Development Team for their support.

Finally, I would like to thank my family - my brother, my mother and my father - for their love, trust, understanding and every kind of support not only throughout my thesis but also throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AE	Application Engineering
AFP	Adjusted Function Points
DE	Domain Engineering
DET	Data Element Type
FB	Feature Based
FIG	Feature Inclusion Graph
FSM	Finite State Machine
IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Point Users Group
EI	External Input
EIF	External Interface File
EO	External Output
EQ	External Inquiry
FTR	File Type Reference
ILF	Internal Logical File
MBT	Model Based Testing
METU	Middle East Technical University
NOD	Number of Defects
NTC	Number of Test Cases
NTCE	Number of Test Case Execution
PDF	Probability Density Function
RET	Record Element Type
ROI	Return On Investment
SC	Software Component
SP	Software Product
SPL	Software Product Line
SPLE	Software Product Line Engineering
SRS	Software Requirement Specification
STD	System Test Descriptions
TADES	Teknik Ateş Destek Sistemleri
TC	Test Coverage
TE	Testing Effectiveness
TEF	Test Environment Complexity Factor
TTP	Test Team Productivity
UFP	Unadjusted Function Points
UFPC	Unadjusted Function Point Count
V	Variant
VP	Variation Point

GLOSSARY OF TERMS

Function Point	A unit of software size measurement that expresses the amount of business functionality an information system provides to a user.
Test Coverage	A measure of the proportion of a program exercised by a test suite, usually expressed as a percentage of the total software size in lines of code.
Testing Effectiveness	The number of bugs found in a testing stage divided by the total number of bugs found throughout the development process.
Testing Efficiency	The ratio of cost savings incurred by testing to the cost of testing.
Variant	A specific choice or extension that completes a system definition by specializing it at a specific variation point.
Variation Point	A location within a (software) system at which a specializing variant must be applied. Variation points indicate where a system is yet abstract and incomplete and an attached variant is required before the system becomes specialized and usable.

CHAPTER 1

INTRODUCTION

In the industry, companies have provided standardized products with the same features to the customer for a while. However, customers' demands change from one person to another. For example, customers may either want to play computer games or want to write documents on the computer. However, they do not want to pay higher values for unused product features. The industry has to satisfy this mass customization for customers [7]. On the other hand, platform based development and mass customization have to be integrated for fast delivery of a higher number of products due to increase on the cost of individualized products. In addition, software products have to be individualized with the features which are dependent to the customer needs. Platform based development and mass customization are integrated in the software industry. The term used for this integration is Software Product Line (SPL).

SPL is the offered technique to decrease the total development cost by the help of reusability and variability. On the other hand, it is a fact that the increasing number of different variations results in increasing cost of the verification and validation process. Since each process of software development is verified via the internal verification process of the company which is mostly used as V model [2], software testing cost can be thought as 50 percent of total software development cost. Software testing also contains validation of the customer specifications before the delivery of the product. Furthermore, higher testing effectiveness (TE) values at each phases of software testing are critical for the elimination of risks before the delivery.

Reusability of software components during the development is the main concern of Software Product Line Engineering (SPLE). In other words, common components are developed during Domain Engineering (DE), and then they are reused in Application Engineering (AE). In the literature, SPL testing can be done more effectively and efficiently by the help of reusability. Total testing cost can be decreased as a result of reusability. There are studies about the integration of DE and AE. The integrated technique is named as W model of software development [11]. It is simply connecting DE and AE's V model development. Reusability of testing artifacts is the connection between DE and AE. Reusability techniques are discussed in the articles [16, 18, 20, 21]. Basically, these techniques suggest reusing scenario, feature, test model and requirements based testing artifacts on AE that are created on DE.

Requirements based testing artifacts creation is used in SPL system tests of Aselsan. However, it is a disadvantage that each test case is only linked to single requirement to satisfy traceability between DE and AE test cases. Thus, the number of test cases (NTC) is equal to the number of requirements. Scenario based testing is not suitable for this reusability technique. On the other hand, model based reusability is the creation of testing artifacts by connecting test cases to a model. Then, a model constraint is defined to calculate NTC from the number of requirements. NTC is a metric for the calculation of testing effort. This calculation method is defined in part 2.2.1. If NTC is higher, testing effort can be much higher than expected. In this study, testing artifacts creation methods are evaluated from the view of NTC calculation, and then they are compared in terms of NTC.

Another subject of this study is about SPL projects' testing effort estimation. SPL testing effort estimation can be done for two different types of testing methods. Software products can be verified by product focused or infrastructure based testing methods. Product focused testing method is a pure application strategy defined in article [7]. In fact, software products are only verified on AE. However, software components can be verified on DE or AE. It is infrastructure based testing strategy. In the present study, testing effort estimation of these testing methods will be evaluated depending on the studies offered in the articles [28, 32, 33, 41].

At Aselsan, infrastructure based testing method is used for the verification of software components. Although complex test environment and much more testing effort are needed for the execution of test cases, it may be thought as a future investment of testing. Two testing methods are compared in terms of testing effort in this study by using testing effort calculation methods [28, 32, 33, 41]. By using the obtained results, test manager of the company could make a decision about these testing methods in SPL testing.

The other concern for SPL testing is the increasing number of software component combinations. In fact, higher number of possible products would occur because of the optional features specified on DE. If testing phase starts after the variation points are bound to software products, there are multiple numbers of combinations for the verification. Testing phase has to start before the variability binding due to this fact. In the literature, there are studies dealing with the early verification techniques of variation points [29, 30]. Design for testability is also a supporting point for early verification techniques [15, 16, 17].

At Aselsan, there is requirement based verification for the common software components. However, there is no study about linking the requirements to the feature tree during the development. Feature tree is useful to generate test case sequences because dependencies between the requirements and features cannot be easily seen by only using software requirement specification (SRS) document. Then, a testing algorithm is necessary due to the increasing number of variation points. The offered testing algorithm on this study is Feature Inclusion Graph (FIG) Basis Path method [29]. A case study is applied on a software component developed at Aselsan for the generation of testing algorithm using FIG basis path method. Then, a comparison is done in terms of testing effectiveness (TE) between the FIG used testing approach and the traditional testing approach.

Companies try to handle the risks on SPL development. Therefore, they try to satisfy the early validation [7] criterion for the testing purposes. Platform tests are not easy to be executed on DE because of the absent variants [7] and an increasing number of combinations. Therefore, Pohl offers that a reference product has to be selected on platform tests to decrease the testing effort [3]. Thus, possible number of software products can be grouped for platform tests at DE. Redundancy formula is used for these groupings and then a project is chosen as a reference product on each group [21].

Platform tests are defined as system tests during DE [11]. Products' system requirements are only verified during the system tests of AE at Aselsan. Then it is risky from the perspective of early validation criterion [7]. Although it does not provide 100 percent of requirement coverage, testing effort would be decreased by the selection of a reference application. SPL products are similar to each other because they are constructed with the similar software components. Then, it is possible to find the same defect more than once during the testing of similar products. On this study, redundancy values are used for the groupings of software products. Afterwards, a reference software product is selected on each group and TE values of platform tests for each software product are evaluated.

The outline of this thesis is as follows: Chapter 2 provides a review of the literature on SPL testing and it divides the subject of SPL testing into four groups, namely, testing process, assets & criteria in SPL, testing metrics & effectiveness, testing effort calculation on testing methods in SPL and reusability of testing artifacts in SPL. It gives background information with the literature coverage about SPL testing on defined topics. Chapter 3 specifies SPL testing process of Aselsan and then it introduces the case studies that would be performed on the projects of Aselsan. In Chapter 4, firstly we propose generating test cases, not from requirements as the traditional approach in Aselsan entails, but from the feature model of the software. This way, test cases' application order may also be based on the feature model, and consequently TE may be increased. Then, testing effort in Aselsan's SPL projects are estimated on two types of testing methods. Thus a method is offered for testing effort estimation with the literature coverage. Next, reusability of test cases is issued on

Aselsan's projects and NTC comparison is done between two proposed techniques, and finally how selection of a reference project for the platform tests affects TE will be studied. Chapter 5 is a general review of the reported work, and it concludes the document.

CHAPTER 2

SOFTWARE PRODUCT LINE TESTING

Effective testing strategies are essential for SPL testing. Although the effort required for software development decreases, testing effort increases due to the complexity of the combinations. Early literature on SPL does not deal with the testing process [9]. When literature survey studies on SPL testing are covered, SPL testing can be issued in different categories. They are unit testing, integration testing, functional testing, SPL architecture testing, testing process & effort, test management, test organization and test automation [6, 9]. This chapter does not contain the literature search about the topics of unit testing, integration testing and test automation. SPL testing is divided into four categories at this chapter. These categories are testing process, assets & criteria in SPL, testing metrics & effectiveness, testing effort calculation on testing methods in SPL and reusability of testing artifacts in SPL. These topics will be considered in detail.

2.1 Testing process, assets & criteria in SPL

Mc Gregor [10] associates the testing process of SPL with the product line architecture. Whole products that will be constructed using SPL use the same software architecture. He divides the testing assets into 5 categories. These categories are test plans, test cases, test results and reports, infrastructure tools and personnel. In addition, test documentation standard is published by the Institute of Electrical and Electronics Engineers (IEEE) [8].

The first part is about the testing process of SPL projects. Then, testing assets, which are used on SPL testing process, will be considered. The last part is about the testing criteria used for SPL testing method evaluation.

2.1.1 Testing process

Testing starts with the unit tests. Unit tests concentrate on the software code. They are mostly performed by developers. White box testing is a part of unit tests. In addition, integration tests are divided into two categories. They are software and system integration tests. A system may contain many software products. A combination of software products can be set and tested during system integration tests. In fact, it is an early testing phase of software development. However, whole system development may not be completed at this phase. Then, the last phases of testing are the system and acceptance tests. By the synthesis of the customer requirements, validation test cases are written. Validation tests are used to examine whether the system responds to the user needs or not. They are acceptance test cases. System tests are applied on the real software configuration. A system may contain a number of software products. The interactions between these software products are verified using real software products. Lower level testing like integration or unit tests can be done by external simulators of the software components or products [2]. V model of software development is seen on Figure 2.1.

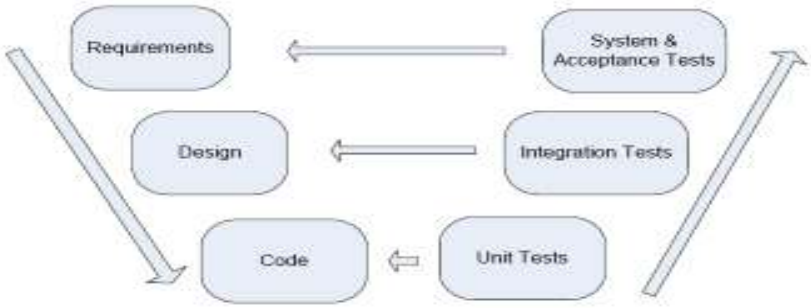


Figure 2.1 - V model of Software Development [2]

Testing process of SPLE is divided into two categories, namely DE and AE. Thus, each divided process has its own V model. However, thinking the testing process in two different V models is not useful for the reusability. Instead of division, the article connects these two V models and then names the testing process as W model [11]. The main concept is the reusability of testing artifacts obtaining from DE and then using them on AE. The testing process is seen on Figure 2.2.

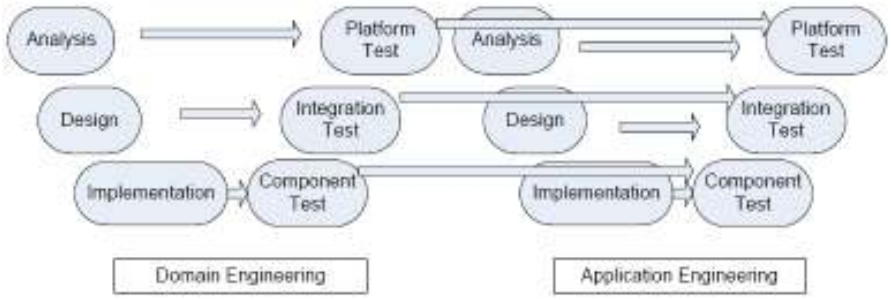


Figure 2.2 - W model of SPL development [11]

W model seen on Figure 2.2 is the modified version of V model according to SPLE. W model's main objective is the reusability of test cases and test scripts written on platform, integration and component tests [11].

2.1.2 Testing assets

Test Plan: Test plan is the starting point of testing stage. It describes test cases to be performed, assigns priorities to the test cases, and determines test resources and tools. Resource allocation and resource consumption are determined for SPL testing. A rough estimation of resource consumption has to be specified for the common and variable parts due to the difference on variability between SPLE and single system engineering. It has to contain which common and variable test cases have to be performed. Test plan is also written to assign priorities on test cases. NTC, which are planned to run, would have higher priority at DE but lower priority at AE.

A template test plan can be created at DE [5]. Afterwards, it may be customized for a product. Some parts of the test plan outline specified at IEEE documentation [8] are customized for SPL Testing at the article [5]. The generation of the test plan template on agile based SPL development is also defined on the article [4].

Test plan is useful for the prioritization of test cases. It is a plan for normal test executions and regression tests. Decision support for the selection of test cases on regression testing is another crucial criterion for test management because the personnel and time are limited during the work for a company. If there is an overhead on regression tests, it will be a waste of time [31]. Therefore, there has to be a criterion to select proper regression test cases. Possible criteria are:

1) The changes on software and side effects of this change have to be verified.
2) The prioritization techniques can be used to determine the test cases on the regression tests.
3) Redundancy has to be minimized by the help of reduction techniques.

Test Case: A single test case contains the objective, input data, test case scenario and fail/pass criteria. Test case would contain one or more test case scenarios. Division of test cases into mandatory and variable parts is necessary for SPLE due to variability testing. Test data also differ on different variations. Example test cases, written for the variation points at DE, are seen on Table 4.3 [5]. On the other hand, environmental needs may differ on the variations because each variation could run on different hardware or operating systems. Redundancy between test cases is also significant to prevent overhead. Redundancy of test cases can be divided into three categories [31]. A test case is redundant,

1) If it is executed twice and has the same result with the previous test.
2) If it covers the requirements which are covered with other test cases.
3) If it covers the same requirement item twice.

Test Results and Report: It is a document that reports the results of the performed testing tasks. It has a limited useful lifetime and provides found defects to development teams in detail. In fact, the report is propagated to the product development team to specify the source of defects for the existing components. In addition, they are also provided to the component engineering team from the product team with contexts for the failures that occur [10].

Infrastructure and tools: Testers use an automation tool for the testing process. These tools are used to save time. DE component engineers generate test scripts. These test scripts are used by the product engineers on AE.

Personnel: The personnel needed for the testing process are divided into 3 categories. These are engineers who write the test cases, engineers who apply the test cases and engineers who write testing tools for the testing process [10]. Personnel working on SPLE development are categorized under two main activities in the article from the perspective of testing [5]. These activities are core asset and product development. On the core asset development, software engineers have to provide proper designs for testability because of the necessity of the core assets' early verification. Then, they have integration test feedbacks that come from the product development. It is useful for the integration of core assets [5].

DE process components are reused at AE [1]. The purposes of DE process components can be seen on Table 2.1. DE process components are used to define reusability during the work on the next chapters.

Table 2.1 - Reusability of DE results on AE [1]

DE process component	**Main purpose**
Domain Analysis	Reusable requirements defined for AE in the domain
Domain Design	Establishing a common architecture that is common for software products
Domain Implementation	Implementing reusable assets.

Mc Gregor defines the organization of SPL testing team by dividing the members into two categories [10]. They are the personnel working for the common and product specific components. Test cases for common requirements and test scripts for common features can be used by the test personnel working for the product specific components.

2.1.3 Testing criteria

There are SPL testing criteria which are defined to give an idea about testing on the development phase. Although some of the criteria are supported by a testing strategy, they could not be satisfied by other strategies. Criteria are used to evaluate testing strategies [4, 7]. These criteria are time to create testing artifacts, absent variants, early validation, learning effort, overhead and testability.

Time to Create Testing Artifacts

Time to create testing artifacts depends on the difficulty of creating them and complexity of the software components which is related to variability.

Time to create testing artifacts is a criterion for the time prediction of the whole testing process because it is the most time consuming activity for a test engineer. NTC is used on the evaluation of software project's testing effort. *Time to create testing artifacts* is the estimation of the overall time required for creating test artifacts in DE and AE [7]. There are some techniques to decrease the time required for the creation of testing artifacts [16, 26, 18, 19, and 20]. Each technique will be discussed on part 2.4.

Absent variant

It is the change of the variability between the start of AE and delivery of the product. Each Variation Point (VP) of the software component cannot be specified during DE because the product is mostly individualized according to the customer demands before the delivery. Adding new application specific software components on AE or testing the common software components on AE are used for the elimination of absent variants' effects.

Early validation

It is the comparison of the defects found at DE and AE. Verification of software components at earlier phases of development reduces the cost of SPL development. Found defect on common software components at AE is to say that the same defect also appears on the previously delivered software products [5]. Elimination of found defects on the previously delivered software products may result an increase on the cost of SPL development.

Learning Effort

The main difference between SPL and single system testing is the variability. Test engineer has to deal with the variability for creating and executing test cases of SPL. Although single system test engineer writes test cases for a single product, SPL test engineer has to deal with a combination of products. If the company uses infrastructure based testing method, the interfaces of the software component have to be tested by using a stub before the integration [5]. Then, test engineer's learning effort is higher because of the required knowledge to simulate the interfaces of software components.

Overhead

Overhead occurs when an unused component is tested, or no defects are found by the execution of test cases more than once. Variability is concerned during DE. Dealing with a non-delivery product during the creation of the test cases would result an overhead. Overhead can also occurs on

regression testing. Regression tests are done to verify whether the defects on the software are solved or not. On the other hand, execution of test cases many times on regression tests would be a time consuming activity. Thus, required testing effort increases because of the repeated test case execution. There are test automation tools for component, integration and unit tests [22, 27]. Test automation is a preferable technique to eliminate overhead. Regression testing decision support is a guide for software test engineer to decrease the total time required for regression tests [31].

Testability

Testability is defined as "the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met" [15]. Software testing requires an efficient and effective testing model. Test Engineers' effort is connected to the software architecture. Regarding only testing models and efficient testing techniques cannot be helpful for software testing. Testing has to be considered from the starting point of software architecture. Defects on a common component have to be detected earlier on SPL testing. If it is not determined, the defect would propagate to each software product because the common component, which contains the defect, is a part of each software product. Mc Gregor defines the probability of a defect on SPL by the formula [16].

$$\exp ectedNumFailures = \sum_{i=1}^{n_p} Pi(d) * (nc_i * x_i)$$

Where n_p denotes the number of products, $Pi(d)$ is the probability of a defect in the component causing a fault, nc_i is the number of copies and x_i is the number of executions for a given product.

2.2 Testing metrics & effectiveness

TE is a quality metric of a testing environment which is related to a testing phase [38]. It is directly correlated to the metric of defects by detection phase. This metric can only be obtained at the end of the study. On the other hand, test manager wants to have an idea about TE value of the testing environment. He or she would have an idea about the coverage of requirements from the start of work. Malaiya et al [36] gives a relationship between TE and test coverage (TC) with a mathematical formula depending on experimental results. In this section, testing metrics will be reviewed and then mathematical formulas for the calculation of TE will be presented.

2.2.1 Testing metrics

This part is divided into 3 categories. They are defect, requirements coverage and redundancy metrics.

Defect metrics

The article divides defect metrics into 6 categories [4]. For the calculation of TE, two types of defects' metrics are used. The other metrics related to defects can be seen at Appendix D.

Defects by Detection Phase: This metric helps to specify the most effective testing process of SPL stages. If the number of defects (NOD) found on DE are higher than NOD found on AE, DE testing is much more effective than AE testing.

Defects by Priority: It is the classification of defects by their priorities. Defects are classified on the priority states of 'High', 'Medium' or 'Low'.

Requirements' coverage metric

A test suite must contain at least one test case per requirement to satisfy requirements coverage. In other words, when test case is executed, it causes the requirement to be met [39]. Then the formula for the requirements coverage will be as in the following.

RC = Number of requirements linked to a test case / Total number of requirements.

The relationship between probability of defects detection and coverage is defined in [36] as:

$$\Pr\{\text{det}\quad ection\quad |\text{cov}\quad erage\quad = c\} = a_0 * \ln\left[1 + a_1 * \left(e^{a_2 c} - 1\right)\right]$$

Where $a_0 = 10.45$, $a_1 = 3.79 \times 10^{-3}$, $a_2 = 3$, c denotes the coverage from 0 to 1 [36].

Redundancy metric

Similarity between software products can be calculated by the redundancy formula defined in the article [21]. It gives an idea on how software products' testing differ from each other. If the redundancy value is one, two products are same.

If a product (pi) is constructed, Fpi is the features of the product and $CFpi$ is a function:

$$CF\quad = \{\{\ fi\} \in N \left| \begin{array}{l} \forall\{fi\} \in N \wedge fj \rightarrow fi, \forall k \in N, \lambda(fj) = andk \quad \cup \\ \{fl\} \in N \left| requires \quad (fi, fl) = true \right. \end{array} \right.$$

The formula of redundancy between two products is calculated as: [21]

$$r(pi, pj) = \frac{card\ ((Fpi\ -\ CFpi\ ) \cap (Fpj\ -\ CFpj\ ))}{card\ ((Fpi\ -\ CFpi\ ) \cup (Fpj\ -\ CFpj\ ))}$$

Test Team Productivity (TTP)

Productivity is defined as to complete the objective in a given unit of time. TTP is related to the number of staff and available personnel for testing [41, 42]. To estimate TTP, rank and proficiency of testers have to be considered. A Tester Rank Model is used for this estimation, and it can be seen on Figure 2.3 [41, 42].

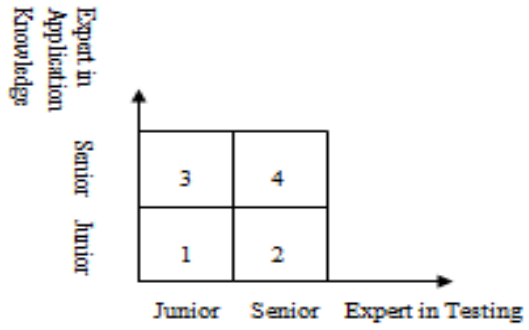


Figure 2.3 – Tester Rank Model [41]

Testers' behavior is estimated by using tester rank model. If the rank of test team is higher, lower the number of testers are needed. The formula for the calculation of TTP is in the following, where T_i shows testers and R_{ij} is the corresponding rank obtained from the rank model [41, 42].

$$TTP = \sum_{i=1}^{n} T_i \times \sum_{j=1}^{4} R_{ij}$$

Testing Effort

Testing effort metric is directly related to NTC for requirement based testing because software test engineer spends time for planning, writing and executing test cases. Then TTP and NTC multiplication results testing effort in man-hours [41].

Testing Effort = NTC x TTP (man-hours)

2.2.2 Testing effectiveness

Measurement of TE is related to number of found defects. 'Defects by priority' metric is the classification of defects according to their importance. On the article, defects prioritization is specified as in three levels [4]. They are 'High', 'Medium' and 'Low'. Then, TE is calculated with the following formula on the article [23].

TE = (Weighted defects reported by testers / Total weighted defects reported) x100

TE value changes from one testing stage to another. An example calculation of TE value can be seen on Table 2.2. TE of each testing level is evaluated with the available defects on the software. For example, TE value of integration tests is calculated as weighted defects reported for integration tests are divided to the sum of weighted defects reported for integration testing, system testing and defects come from customer. In other words, defects found from functional testing do not affect TE value of integration testing. An example calculation of TE can be found on Table 2.2.

Table 2.2 – Example of TE calculation with the found weighted defects [23]

Defects by Priority	Functional Testing	Integration Testing	System Testing	Defects from Customer
High	3	2	1	2
Medium	2	1	1	1
Low	2	0	0	1
Total NOD	7	3	2	4
TE	7/16=%44	3/9=%33	2/6=%33	-

If weighted NOD cannot be measured, other TE calculation method is related to the requirements coverage metric. From the start of work, software test engineer would calculate the software product's test coverage metric value by using the number of test cases which are linked requirements. The offered formula for the calculation of TE on the article is the following [36].

$$TE = \frac{\int_{cx}^{1} \Pr\{\det ection \mid \mathrm{cov} = c\} * \Pr\{\mathrm{cov} = c\} * dc}{\int_{cx}^{1} \Pr\{\mathrm{cov} = c\} * dc}$$

2.3 Testing effort calculation on testing methods in SPL

Although the companies apply SPL technique to reduce the cost of software development, testing effort increases because of the complexity and increasing number of combinations [9]. Resource allocation is crucial for a test manager to calculate the cost of software testing. Therefore, test manager has to find a way to decrease the effort on SPL testing. The article defines a method for the estimation of product family testing's effort [12]. It divides the effort of SPL Testing into 3 categories which are defined on Figure 2.4.

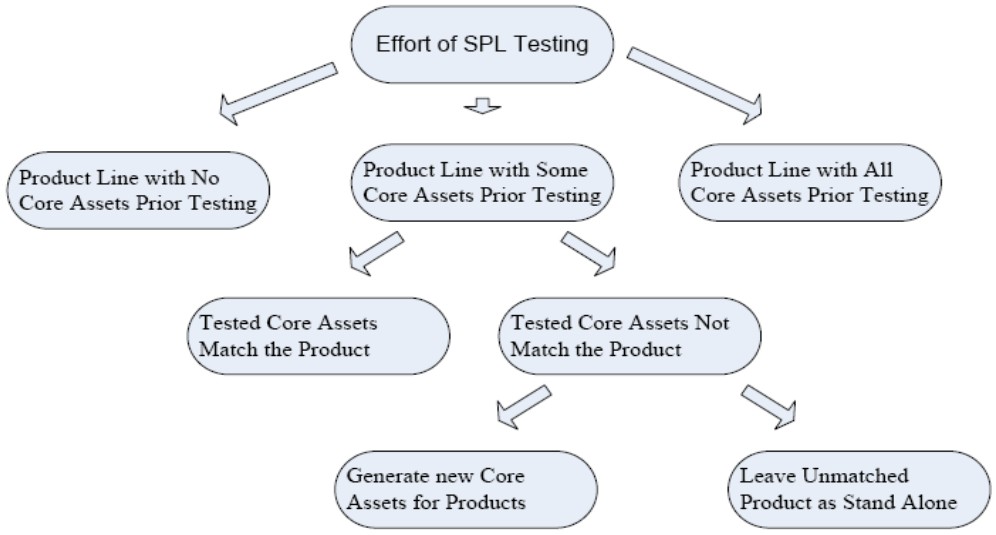


Figure 2.4 - Effort estimation of SPL testing [12]

The article defines a formula for testing effort estimation [12]. Testing effort is calculated by using the defined variables on Figure 2.4. SPL testing effort's formula is in the following.

$$E_{splt} = E_{org} + \sum_{i=1}^{N_{core\_assets}} (E_{components(i)} + E_{core\_assets\_build(i)} + E_{\sup port(i)} + E_{ma\,int(i)}) + \sum_{i=1}^{N_{products}} (E_{unique(pi)} + E_{reuse(pi)})$$

The definitions of the variables in the formula:

E_{org} **:** The effort to an organization of adopting the product line approach for its products

$E_{components}$ **:** Effort of components' test

$E_{core\_assets\_build}$ **:** Effort of building core assets

$E_{\sup port}$ **:** Effort of supporting the generic software testing architecture, documentation, and test infrastructure

$E_{ma\,\mathrm{int}}$ **:** Effort of repository maintenance

E_{unique} **:** Effort to generate unique software testing artifact that is not based on a product line

E_{reuse} **:** Effort to reuse core assets in a core asset base

Buckle proposes a formula for return on investment (ROI) [13]. The meaning of ROI is the calculation of the process efficiency.

ROI is defined by the formula [12]:

$$ROI = \frac{GainofInvestment - CostofInvestment}{CostofInvestment}$$

In the industry, there is a case study applied for two strategies, which are product focused and infrastructure based [28]. The ***product focused strategy*** deals with the product's architecture. There is not a testing method offered for software components. The integrated software components are thought as single software. Then, the components are verified together. A component can be reused on large number of software products. When a defect occurs on a component, and it is not discovered, it spreads over the software products. The other disadvantage of product focused testing strategy is the execution of test cases many times which is named as overhead in the literature [7]. The article defines NTCE on SPLE with the following formula [5].

$$NOTCE = \sum_{i=1}^{n_p} (nc_i * x_i)$$

Where; n_p is the number of products; nc_i is the number of copies. Then, x_i is the number of executions for a given product.

Infrastructure based strategy is the division of the software product into software components and then testing them. The common components are tested individually. After the individual testing, the product is assumed to be working correctly because requirements are verified. Therefore, the propagation of defects resulting from the common software component is blocked earlier. The disadvantage of this strategy is testing different feature variations of the unintegrated software components. Software architecture of the components has to be designed by dealing on testability. A stub would be used to test the interfaces of the software component.

SPLE is a technique that is used for the development of resembling software products. ***Product focused strategy*** of SPLE is used to test similar software products. However, if this testing technique is selected, test cases are generated and then executed for the common software components over and over again. Common software components are used on software products. Then, found defects resulting from common components on the software decrease [28]. The company has to give a decision on the selection of the appropriate testing strategy by analyzing TE and testing effort results.

If infrastructure based testing strategy is applied, regression test cases of a common software component are selected by the evaluation of changes on software component. However, these changes would result in new defects only on this component [31]. In fact, there is no need to cover

other unchanged software components' requirements. On the other hand, product focused testing approach deals with software product's requirements. A change on a common software component would result in new defects on software product. In fact, number of regression test cases has to be higher than the number of infrastructure based testing approach's regression test cases.

Testing effort and future investment are the main criteria for a test manager to choose proper testing strategy. Infrastructure based testing approach is preferable for the future investment because testing personnel verifies software components before the integration. It can be used on many software products. On the other hand, there are no verified software components before the integration when product focused testing strategy is chosen. There is a method offered to calculate two strategies' testing effort [28].

Testing effort estimations for each product are summed up to calculate total estimated testing effort for the product focused testing strategy.

$$C_{product\_focused} = N * C_{prod}$$

Infrastructure based testing firstly focuses on reusable components. Reusable components are tested on DE. Product specific and adaptation to product line components are tested on AE phase.

$$C_{\inf rastructure\_based} = C_{reusable} + N * (C_{adapt} + C_{product\_specific})$$

Saved testing cost can be calculated with the following formula.

$$C_{saved} = C_{product\_focused} - C_{\inf rastructure\_focused}$$

Testing effort estimation is necessary for a test manager because test manager has to have a prediction on the total cost of testing. Resource allocation planning on software testing is necessary. In addition, test manager makes a decision on a testing method from the starting point of testing process. Testing method selection would change depending on the number of personnel. Thus, applying a testing method without gained experience is risky for a company. The article defines a method for the rough estimation of NTC with the knowledge of number of function points [33]. In fact, NTC is equal to the total number of function points with a power of 1.2. At this study, number of FPs is calculated by using International Function Point Users Group (IFPUG) method [40]. In short, testing effort would also be calculated by the knowledge of NTC, Testing Environment Complexity Factor (TEF) and TTP. Testing effort estimation method can be found on Appendix A.

2.4 Reusability of testing artifacts in SPL

Domain requirements engineering has an output of domain requirements artifacts and variability model. Testing mostly depends on software requirements [7]. Domain requirements contain the variable and common software requirements. Creation of the testing assets on AE will be easy by regarding the variability on DE. There are four approaches that are related to the reusability of test cases during AE.

Clementine et al considers creation of reusable test case scenarios [18]. In fact, test scenarios are created during DE and used on AE. Hartman defines model-based testing [19]. However, test cases are only derived on AE. There is no reusability between DE and AE. Then the last approach for test case generation is the creation of the model based reusable test cases [20].

This part consists of two sub-sections, which are requirements based reusability and model based reusability.

2.4.1 Requirements based reusability

According to Mc Gregor, a test engineer has to create reusable test cases during DE by using the variability [16]. These test cases are derived from the natural requirements. Then, test cases are used on AE by using the traceability links. Simple model is seen on Figure 2.5.

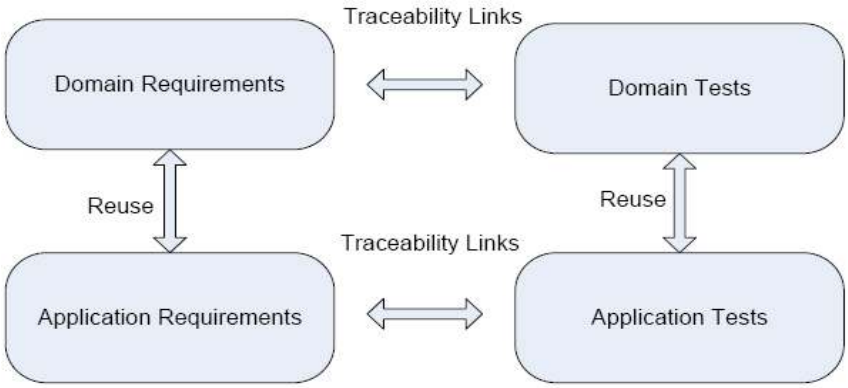


Figure 2.5 - AE testing artifacts creation [16]

It is a useful technique for system and acceptance tests. A test suite must contain at most one test case per requirement because of traceability restriction from DE to AE. Then, NTC will be equal to the number requirements. Scenario based testing is not suitable for requirements based reusability.

2.4.2 Model based reusability

During the domain analysis, use cases are defined with activity diagrams. In the article, test cases are written for the whole variants on DE and then linked to use cases. Test cases are modified depending on the selected variants of the application during the transport of test cases from DE to AE [20]. In other words, test cases obtained from DE are reused with a few modifications on AE.

Test model is created by the help of use cases specified on the requirements. Then, test cases are created with the usage of this testing model. For each activity, there is traceability between DE and AE. The model is seen on Figure 2.6.

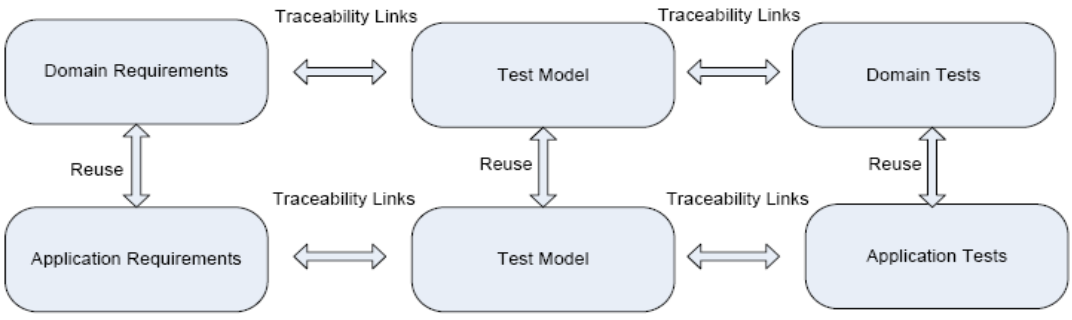


Figure 2.6 - Test model usage for SPL [20]

Test model defined on Figure 2.6 can also be thought as a Finite State Machine (FSM) instead of activity diagram [24]. It is mostly used technique to write test cases easily because it defines the software in more detail. Since there is a difference between single system engineering and SPL

development, variability has to be concerned during the creation of FSM model of the software product on domain analysis. Test model on AE changes depending on the selected variants.

Perrouin et al offers a feature based test case generation method [21]. Feature model obtained from domain analysis is used for the generation of test cases. Using this model, possible number of software products can be determined for platform tests at DE. However, dependencies between features have to be specified on this feature model. If there are no specified dependency details between features, feature based testing is not a preferable technique for test case generation. Cabral et al defines FIG Basis Path algorithm for the generation of test cases [30]. It is also a test sequence generation method using the feature model. Testing path starts with the mandatory features and then continues with test cases of variable features. After mandatory features' verification, bound variants are added to the testing path for each variation point. Testing algorithm of FIG Basis Path method can be seen on Figure 2.7.

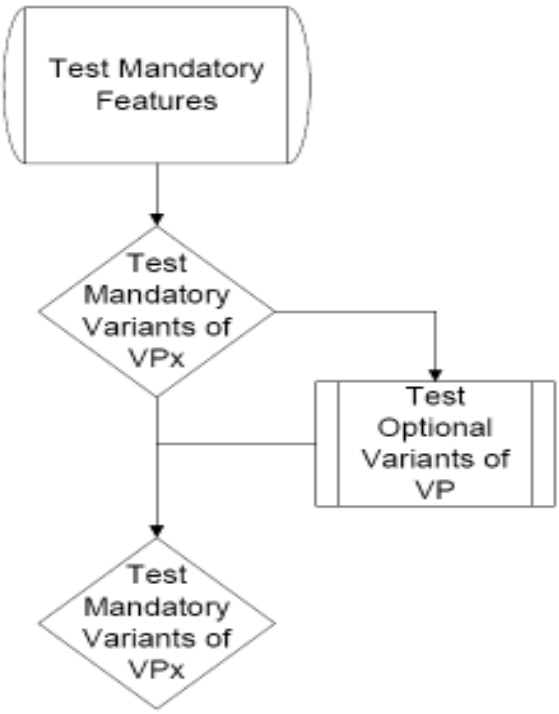


Figure 2.7 - FIG Basis Path Method [30]

Pohl et al offers creation of test case scenarios with the available variants [37]. DE test case scenarios are written according to the whole possible variants. When variants are bound to a software product, test case scenarios are selected depending on the bound variants.

CHAPTER 3

EVALUATION OF THE APPLIED SPL TESTING TECHNIQUES IN ASELSAN

In this chapter, we consider the testing techniques applied in Aselsan, from the viewpoint established based on the literature review presented in the previous chapter. We compare the literature and the company's internal development process on the topics of testing process and assets management, test management, testability and reusability of testing artifacts. SPL development work first started in Aselsan on 2009. From the start of the development, DE organizational structure, working personnel, and software development plan were determined. Then, Teknik Ateş Destek Sistemleri (TADES) is started as a development project for future investment. SPL's reference architecture is firstly developed. Aselsan's development model is seen on Figure 3.1.

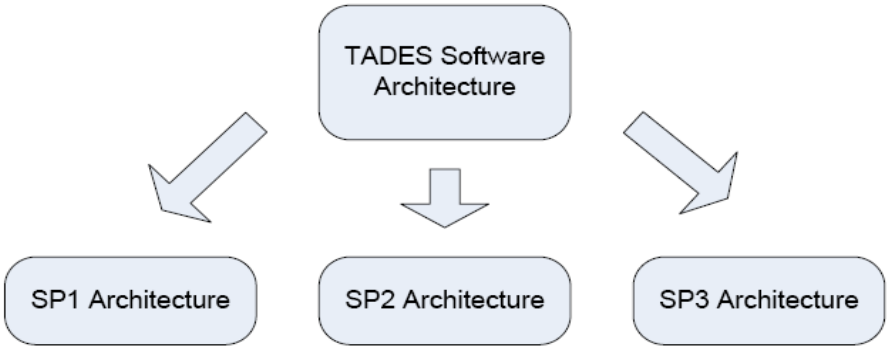


Figure 3.1 - Aselsan's DE SPL architecture development [34]

Figure 3.1 is the software architecture development model of the company. Testing is strongly connected to this development model due to the need for the simulation of external interfaces of software components. The experience and testing materials obtained from DE (TADES) are used for AE projects.

3.1 Testing process & assets management

The internal software development process of Aselsan is the V model development. V model is used for single system software development. V model differs on SPL projects. Each project waits its common software component from TADES. If the project has project specific requirements which would not be useful on the future projects, requirements are satisfied with the development of new product specific software components by AE team.

Software components' integration is not done on DE. The V model development process is applied on TADES up to the software components' requirements' verification. W model applied in ASELSAN entails that AE projects have to be strongly connected to DE project on the development, verification and validation.

Mc Gregor et al [10] defines creation of templates that will be useful on the future projects for test organization and process. Test plan is the start of the testing work. It is written at the start of the project and then it is referenced to software development plan. In this plan, test manager defines the resources to be allocated. Main resource for the software testing is the working personnel.

In the test plan, test manager makes estimation for the required staff. Effort required for testing is estimated based on the total number of requirements. It may be preferable for single system software development. However, testing effort estimation has to be done by considering the reusability of testing artifacts. In the literature, there is a method proposed for effort estimation of SPL testing [12]. Testing effort estimation between infrastructure based and product focused testing methods is covered in part 4.2 by considering the effort estimation formula.

Test management is strongly connected to project management because software product has to be verified and validated during or after work. Test plan estimates the required resources by the help of software development plan. The timeline for software development in Aselsan is shown in Figure 3.2.

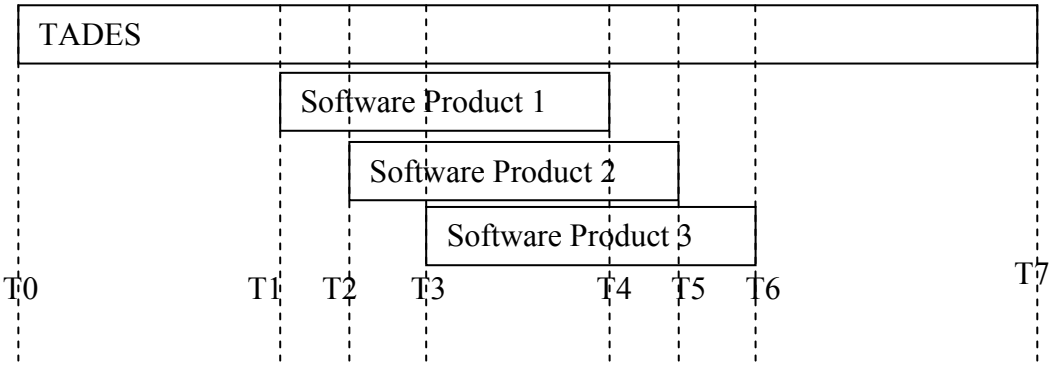


Figure 3.2 - Timeline for SPL projects

TADES is DE project. The other projects are AE development projects that will be delivered to the customer. The projects' starting date would change depending on the Project A's common component delivery. The final delivery dates of the ***Software Product 1, Software Product 2*** and ***Software Product 3*** would also change as a result of the delayed delivery of ***TADES***'s common components.

Test plan is organized for the projects' starting and delivery dates. The template test plan can be prepared for TADES [10]. Then this template test plan is used for other projects.

The other critical issue is the regression test techniques of software components because common software components are not only one time delivered to each project. Configuration management of a software component has to be followed by the software development engineer.

At Aselsan, configuration management of a software component is applied by a new software specification document and a version number. Modifications on the software component have to be tested at regression tests. Thus, software test engineer has to obey the full test case specifications of the software test document in general. It may result in overhead for software test engineer. Decision support techniques are suggested in the article [31] by regarding regression test techniques for test management are proposed.

Redundancy metric is the evaluation of the similarity between software products. If redundancy is higher between two components, they are similar to each other. SPL development's main objective is to deliver similar products. Testing one of similar products gives an idea about the defects on the

other similar product. Therefore, redundancy metric has to be evaluated during the creation of SPL products' test setups. If the redundancy value is higher between two products, testing one of them prevents overhead.

3.2 Testability of variation points

Software development engineers have to design their software components regarding testability because when testability is high for a software component, higher controllability and observability can be obtained on this software component by software test engineer. Variation points can be verified at the early stage of verification.

Software component may be categorized as a common software component or a project specific software component. If software component is common for AE projects, it has to be developed at DE project. Then, software component would have VPs because it is used on different software products. Requirements of the common software component are divided into two parts. They are mandatory and variation point requirements. Then, these requirements are linked to software component's feature tree. Software component's requirements variability can be seen on Figure 3.3.

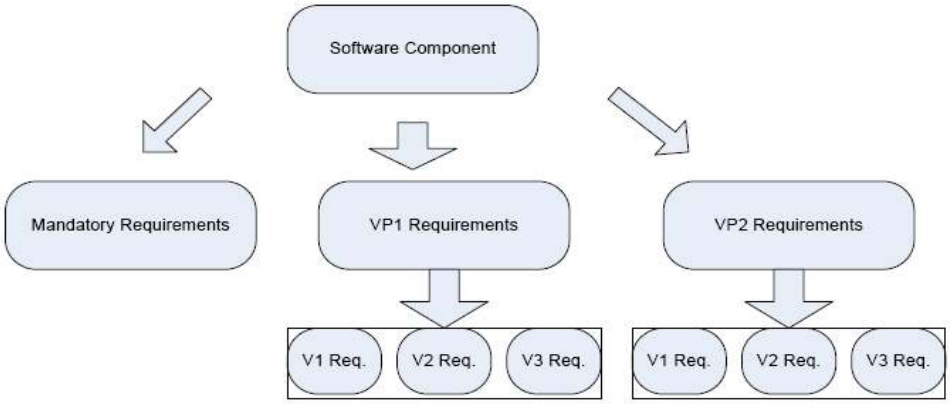


Figure 3.3 – Requirements' variability of software components

Mc Gregor et al [16] say that if defects that are not found on common components during DE would propagate on each project. Thus, higher testability is an objective on the software components at DE. Possible variation points have to be verified at DE. Requirement types defined on Figure 3.3 are bound to software product after the variability binding [29]. At the system level, it may be considered as single software. FIG Basis Path method can be used for the verification of the software component before variability binding [30]. This method simply defines the testing algorithm according to the software component's feature tree. If testability is satisfied by software development engineer, variation points of the software component on different configurations are verified using an external stub.

TADES has many common software components. Each software component has variation points. Therefore, possible number of software products that will be constructed using these software components increases sharply. Requirement's variability view of software components on TADES can be seen on Figure 3.4.

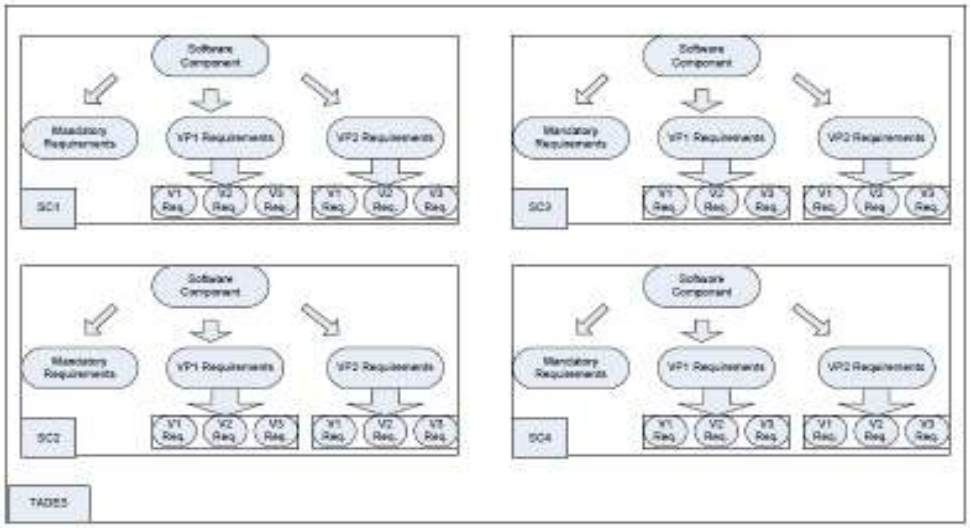


Figure 3.4 – Requirements' variability of TADES

From Figure 3.4, it is seen that each software component has m_i number of possible configurations. Then, if there are k numbers of software components, the formula defines the maximum number of products on the domain. As a result, it is impossible to verify all software products at the system level. TADES's software architecture on software component level has to be designed for testability to prevent the increasing number of product configurations.

Maximum number of products = $\prod_{i=1}^{k} m_i$

Learning effort is higher at this step because interfaces of software components have to be simulated for isolated verification. Gregor et al [5] offers developing stubs for the external interfaces of the software component. This stub uses the same software architecture reference model with other software components. It is directly connected to TADES's software architecture on software component level. Communication structure of an example test setup can be seen on Figure 3.5.

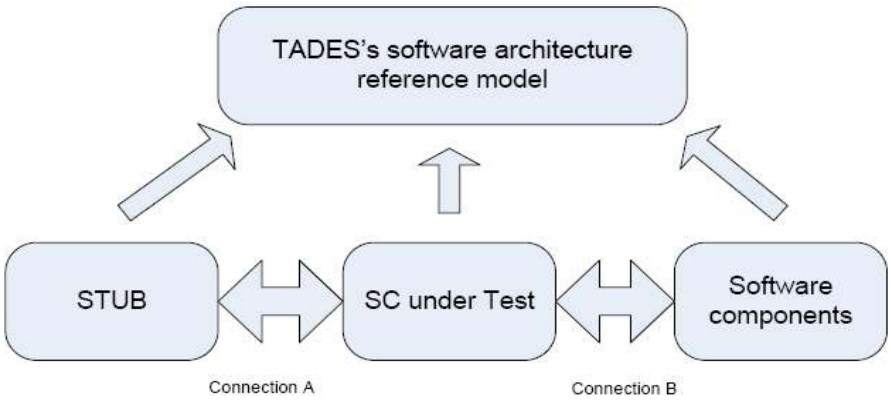


Figure 3.5 - Test setup for common software components

If software component is under test, a stub is used for verification purposes. Then, the simulation of **connection B** is done with **connection A.** As a result, testability is satisfied on component level testing by the software architecture. In Chapter 4, FIG Basis Path method will be discussed for the software component's functional testing and then, TE comparison will be done at this level of testing with FIG and without FIG method.

3.3 Reusability of testing artifacts

Acceptance test, which is the validation of customer requirements, is the final testing stage of a software project. In addition, system tests are the verification of whether the product satisfies system requirements or not. Internal software development process of the company starts with system requirements. Connection between validation and verification can be seen on Figure 3.6.

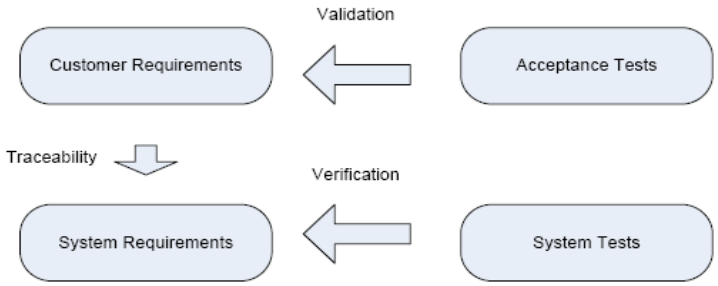


Figure 3.6 – Verification & Validation connection

Number of requirements increases from customer requirements to system requirements because software requirement descriptions are specified in more detail. This requirement process continues for each SPL project. As a result, if the requirements are not categorized as domain and application requirements, then reusability cannot be used for SPL projects. Mc Gregor's idea for the reusability of testing assets is the creation of the reusable test cases on DE [16]. Categorization of system requirements and system test cases applied on Aselsan's SPL projects can be seen on Figure 3.7.

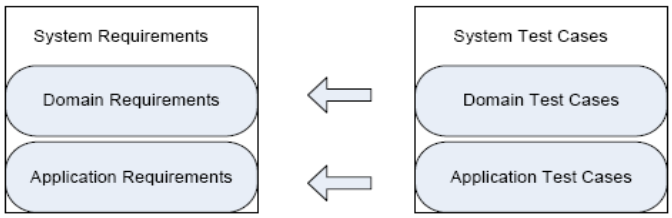


Figure 3.7 - Automatic test case generation

Total NTC for the system tests is calculated with the sum of domain tests and application specific tests. Domain test cases are written at **TADES** and they are ready to be used for **SP1, SP2** and **SP3**. Then, total time required for the creation of the testing artifacts decreases.

Since this technique decreases the time for the creation of testing artifacts, one to one traceability has to be provided by the software test engineer. If two or many requirements are connected to a test case, it may result traceability problem for the future use. Total number of requirements has to be equal to the total NTC for this method. Since total NTC is a metric for the creation of testing artifacts, it may be a disadvantage for time to create testing artifacts criterion. In addition, NTC directly affects testing effort calculation [32].

Feature tree is constructed for each project by the help of domain feature tree [21]. Traceability between domain requirements and domain test cases can be set by the help of a test model [20]. If domain requirements and domain test cases are linked to domain feature tree, software test engineer automatically generates test cases written for the selected features of a software product. Systematic approach for the usage of domain feature tree can be seen on Figure 3.8.

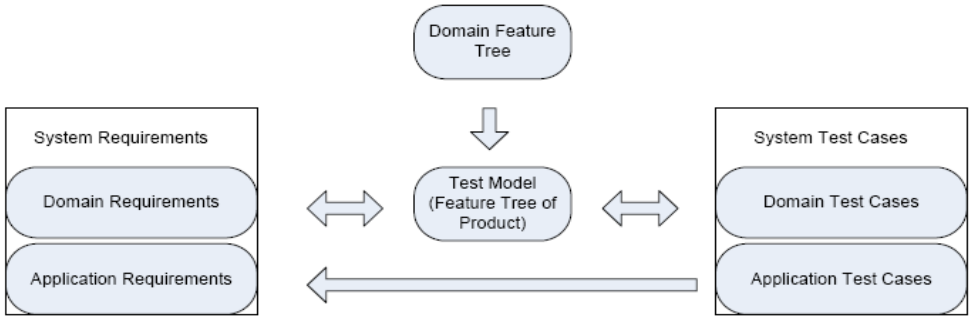


Figure 3.8 - Usage of feature model for test case generation

Comparison between these two reusability methods can be done in terms of NTC at Aselsan. Requirements based test case generation method is applied. On the other hand, there is not a study offered on model based test case generation technique. It may decrease the time required for the creation of testing artifacts.

CHAPTER 4

ASSESSMENT OF THE STUDIES AND DISCUSSION

This chapter evaluates SPL testing work in Aselsan through four different case studies. They are based on TE and testing effort comparisons between offered and applied methods. Chapter 3 gives a general idea about the situation of Aselsan and offers new testing techniques with the materials obtained in Chapter 2. Model based testing (MBT) is offered for test case sequence generation. TE and testing effort are measured within each case study.

The first study is on testing the variation points at the functional testing level of V model by the usage of FIG Basis Path method. Early verification of the software components is an essential criterion for software test engineer. Testing VPs after the variability binding is risky because of the increasing combinations. Therefore, early testing of variation points reduces total testing cost in general. TE will be compared between with FIG and without FIG at the functional testing. Software components will be verified in the scope of infrastructure based testing [28].

The second study is on management of SPL Testing. It is based on the application of infrastructure based and product focused approaches, and comparison of results. Comparison will be done in terms of testing efforts between two different testing approaches using experimental results of Aselsan's SPL projects.

The next study is on the creation of testing artifacts. Reusability is an essential factor to verify or validate software products with little effort. Decreasing the time required for the creation of testing artifacts is necessary because it affects the overall cost of the project. Two testing artifacts creation methods will be compared in terms of NTC using Aselsan's SPL projects.

The last study reported in this chapter is on the calculation of redundancy values among software products. Similar products are grouped for DE platform tests by using the redundancy values as defined in the article [21] to decrease testing effort. Then, redundancy metric obtained between software products will be used for the selection of reference projects. Finally, TE values are estimated using TC values of each software product which are calculated on platform tests.

4.1 Testing VPs of software components and TE results

Aselsan has developed three software products that will be considered within the scope of this study. The names of these products shall be abbreviated as SP1, SP2 and SP3. Abilities of these products resemble each other. In addition, SC1 is used as common software component on these products during the development. It is developed at DE level. Then, it is integrated to these software products at AE level.

When variability binding types on the literature are examined, there is load time variability binding at Aselsan's SPL projects [5]. Software engineer binds variation points to the software component during development. In addition, variation points are linked to configuration file constraints.

SC1 is a software component that has configuration constraints. This software component is used on different software products by the help of these configuration constraints. System level interfaces of software products are seen on Figure 4.1.

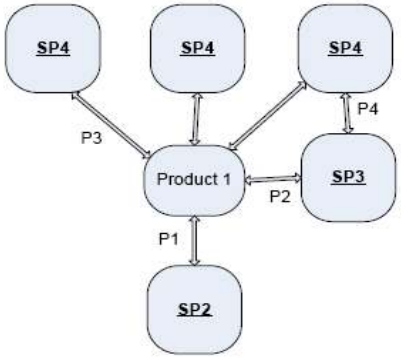


Figure 4.1 - Software products' interfaces

Customer defines software interactions at the system level as on Figure 4.1. In fact, software interaction diagram is extracted from customer requirements. However, it is impossible to see these interactions during DE due to absent variants. Software products that contain SC1 software component are underlined. Verification of the requirements for the software component can be done on the integration level. In fact, software test engineer creates software products' test setups in different combinations. Necessary test setup of SP3 and SP4 has to contain software products of SP3, SP4 and Product1. On the other hand, test setup required for SP2 has to contain SP2, Product1 and SP3. Testing paths for the verification of this software component can be seen on Table 4.1.

Table 4.1 - Required testing paths for software products

Software Product	**Required Testing Path**
SP3	P2-P3-P4
SP4	P2-P3-P4
SP2	P1-P2-P3

Software engineer binds the VPs to the software component and then test engineer verifies the variation points at AE level. It is a pure application strategy that is defined by the article [7]. There are no testing phases defined at DE level. All the test phases are applied during AE level. It is not a preferred technique because it is in contradiction with the early validation testing criterion [7].

There are constraints for software components' VPs. Configuration files are used for variability binding during the functional testing of software components.

FIG Basis Path method is a presented technique for the generation of software component's test cases [30]. In other words, it offers creation of test cases by the help of feature tree. SC1 test cases creation depends on four VPs and mandatory features. These variation points are determined by project staffs. Test cases are written by regarding VPs and mandatory features. UML-based diagrams and SRS document are used during test case creation. Then, test cases are linked to either variants of variation points or mandatory features.

Software component's features are divided into two categories. These features are mandatory and variable features as mentioned on Figure 3.3. Mandatory features are not allowed to be changed from the configuration file. However, VPs can be set to the available variants from the configuration file. Software would have a different behavior during the execution of test cases which are illustrated to VPs because of the changed variant. Therefore, if a new variant is added to the variation point, new test cases have to be generated for the corresponding variant.

During test case generation of SC1 software component, verification of variation points is the main objective for the functional testing. Functional testing is carried out during integration tests [29]. Therefore, some of the external interfaces of software component have to be simulated to execute test cases. This involves writing a stub for the simulation [5].

There are four VPs that can be modified from the configuration file. Feature tree of software component is seen on Figure 4.2. The logical functions and feature modeling technique is gathered from the article [1].

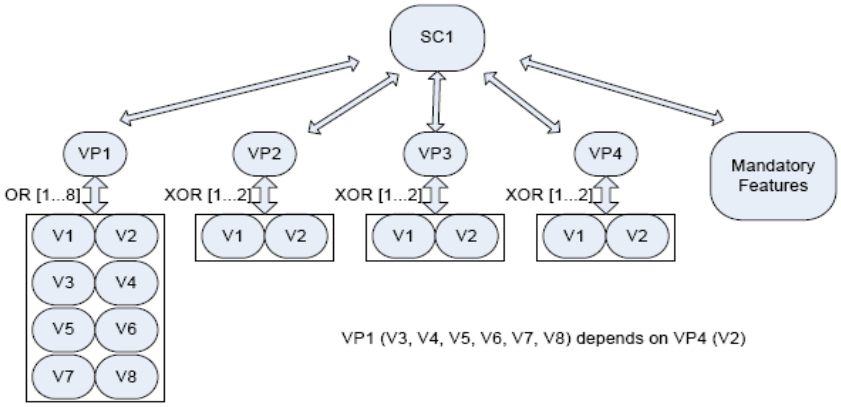


Figure 4.2 - Feature tree of SC1

Requirements' variability is shown on Figure 3.3. Thus, test cases are written regarding the requirements linked to variants. In other words, test cases are linked to variant of a variation point. They are not linked to requirements. SC1's NTC connected to variants is seen on Table 4.2.

Table 4.2 - NTC of VPs' variants

Feature model – Test Paths Traceability	Total NTC
MF	18
VP1-V1	8
VP1-V2	3
VP1-V3	3
VP1-V4	1
VP1-V5	3
VP1-V6	2
VP1-V7	3
VP1-V8	2
VP2-V1	4
VP2-V2	3
VP3-V1	2
VP3-V2	3
VP4-V1	4
VP4-V2	3

FIG Basis Path method is the algorithm for the software component's testing paths. Testing paths are specified and added to basis path set by using breadth first search [30]. Firstly, mandatory features' test cases are covered in series. Then, testing paths are selected for VP1, VP2, VP3 and VP4. These paths are connected to each other and added to basis path set. By using the algorithm, NTC related to the longest path is calculated for SC1:

NTC included on the longest path = 18 (MF) +25 (VP1) +4 (VP2-V1) +3 (VP3-V2) + 3 (VP4-V2) = 53

VPs have to be specified for each test case [5]. Then example test cases, which deal with the variation points, can be seen on Table 4.3.

Table 4.3 - Example test cases written for variation points

Test Case 1 (VP1 : V1)	**Variation points:** VP2-V1, VP3-V2, VP4-V1 **Initial Conditions:** Initial conditions depending on the requirements **Inputs:** Inputs depending on test scenarios **Test Steps** …
Test Case 2 (VP1 : V1)	**Variation points:** VP2-V2, VP3-V2, VP4-V1 **Initial Conditions:** Initial conditions depending on the requirements **Inputs:** Inputs depending on test scenarios **Test Steps** …

The other critical issue is testability. In Chapter 3, TADES's software components' testability is discussed. Simulation of the external interfaces is done using a stub referencing to TADES's software architecture. Constraints used on testing paths are related to the external interfaces of the software product. Input/output relations and state changes of SC1 are configured during the execution of test cases. Then, a stub will be used to check software component's input/output relations. SC1's variation points are bound using the configuration file with the ease of load time variability binding [5]. Then, SC1's inputs can be controlled and also SC1's output responses can be observed using this stub.

Test Case 1 is written to verify VP1-V1 variant on the given variation points. On the other hand, Test Case 2 is also written to verify VP1-V1 variant on another variation points' configuration. FIG Basis Path method firstly offers execution of mandatory features' test cases. Test environment for this software component is seen on Figure 4.3.

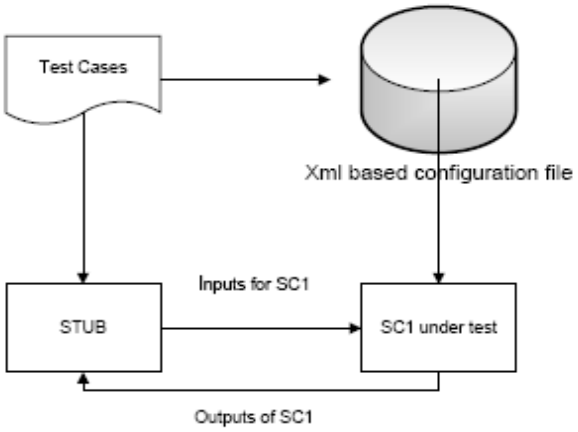


Figure 4.3 - Test setup for SC1 software component

SC1 test cases are executed in the sequence. Test case sequences are offered by the FIG Basis Path method. A test case can be executed more than once because of the required variation point combinations.

If the feature tree of SC1 is analyzed, there are 8 different configurations with the bound variants. In addition, there are dependencies between VPs. There will be a testing path for each combination. Then, total number of necessary test case executions and NOD comparison with and without FIG Basis Path method is also seen on Table 4.4.

Table 4.4 - Test results obtained w FIG and w/o FIG

Feature tree	NTC	NTCE w FIG	NTCE w/o FIG	NOD w FIG	NOD w/o FIG
MF	18	144	18	41	35
VP1-V1	8	64	8	2	2
VP1-V2	3	24	3	3	2
VP1-V3	3	12	3	4	3
VP1-V4	1	4	1	3	2
VP1-V5	3	12	3	5	4
VP1-V6	2	8	2	4	2
VP1-V7	3	12	3	1	0
VP1-V8	2	8	2	5	4
VP2-V1	4	16	4	6	6
VP2-V2	3	12	3	5	5
VP3-V1	2	8	2	2	2
VP3-V2	3	12	3	3	3
VP4-V1	4	16	4	2	2
VP4-V2	3	12	3	0	0
Total	62	364	62	86	72

From Table 4.4, it is seen that NOD increases with the execution of FIG Basis path algorithm. However, number of test case execution (NTCE) increases due to testing on different VP configurations. In other words, if this algorithm is applied, testing effort increases.

TE is calculated as the division of the weighted NOD reported by testers to weighted total NOD found on the software component [23]. This component is delivered to the customer as a part of

different software products. Total NOD reported to the software developer for this software component is 95. This number is the sum of found defects during the integration, functional, system and acceptance test reports.

Weighted defects can be found on Table 4.5. Defects are prioritized by software test engineers using defects by priority metric. This metric is issued at part 2.2.1.

Table 4.5 –TE calculation with the weighted reported defects [23]

Defects by Priority	FT before FIG	FT after FIG	Integration, System, Acceptance Tests, before FIG	Integration, System, Acceptance Tests, after FIG
High	5	7	2	2
Medium	25	29	8	3
Low	42	50	13	4
Total NOD	72	86	23	9
TE	72/95=%75	86/95=%91		

It is seen that TE value increases at FT and number of found defects decreases at integration, system and acceptance tests with FIG basis path method. Then, NOD decreases with the application of FIG basis path method. As a result, although application of this method increases TE value, it increases the testing effort with the increment on NTCE.

4.2 Testing effort comparison between testing methods in SPL

There are two SPL testing methods discussed at part 2.3. These are infrastructure based and product focused testing methods. Product focused testing is verifying software as a single software. On the other hand, infrastructure based strategy is dividing the software products into software components and testing software components individually. The method offered in part 4.1 is infrastructure based testing. Test manager chooses the proper technique considering testing efforts. In this part, a testing effort calculation study will be applied for these methods. Experimental data used during this experiment are SPL projects of Aselsan.

Aselsan's software development process is divided into two parts. System engineering and software engineering are parts of software development. Customer requirements specification document is the starting point of the development. According to customer requirements, system requirements specification document is written. In fact, this document is a reference for the system design document. Software product is divided into components. Each requirement specification specified on system design document is linked to requirements of software components' requirements specification document. This software development process is seen on Figure 4.4.

Figure 4.4 - SPL development requirements' documentation process

In this section, testing effort of infrastructure based and product focused testing methods will be examined for the first three SPL projects of Aselsan. These projects are specified on this study with names of Software Product 1(SP1), Software Product 4(SP4) and Software Product 5(SP5). Testing effort comparison between product focused and infrastructure based testing will be done by using testing effort estimation formulas proposed in [32]. Each software product is constructed with the reusable software components that come from DE. Moreover, reusable and product specific components are adapted to each other during AE.

Infrastructure based testing strategy is applied at part 4.1. Traceability between software component's requirements' specification document and software test specification document is established. Infrastructure based method's testing effort is defined according to the formula [28]:

$$C_{\text{inf}\,rastructure\_focused} = C_{reusable} + N * (C_{adapt} + C_{product\_specific})$$

$C_{reusable}$ is the testing cost estimation during the verification of common software component's requirements.

C_{adapt} is the testing cost estimation during the verification of adaptation on the common software requirements.

$C_{product\_specific}$ is the testing cost estimation during the verification of product specific software component's requirements.

Number of requirements which are defined for each common software component is seen on Table 4.6. Whole requirements are not verified during SP1 testing. Some of them are verified during SP4 or SP5 testing.

Table 4.6 - Reusable software components' requirements

Software Component	Total Number of Requirements
Software Component 0 (SC0)	186
Software Component 2 (SC2)	266
Software Component 3 (SC3)	135
Software Component 4 (SC4)	36
Software Component 5 (SC5)	94
Software Component 6 (SC6)	67
Software Component 7 (SC7)	43
Software Component 8 (SC8)	135
Software Component 9 (SC9)	71
Software Component 10 (SC10)	188
Software Component 1 (SC1)	347
Software Component 11 (SC11)	213
Software Component 12 (SC12)	53
Software Component 13 (SC13)	185
Software Component 14 (SC14)	120
Software Component 15 (SC15)	50
Software Component 16 (SC16)	43
Software Component 17 (SC17)	30
Software Component 18 (SC18)	226

SP1 is the first AE project of Aselsan. Therefore, there are not available reusable tested software components for this project. Then, there is no adaptation of the reusable software components for this project.

Testing effort of a software product is calculated with the help of the number of function points. Total number of function points is calculated with the IFPUG method [40]. Caper et al [33] estimates the NTC at the functional testing with the following formula.

NTC = Number of Function Points^1.2

According to this estimation, infrastructure based testing effort calculation for products of SP1, SP2 and SP5 can be found on Appendix B.

Product focused strategy is another testing method that is defined on the article [28]. Software product is not divided into software components with this testing technique. Thus, the product is thought as single software and it is tested. In fact, interfaces between software components are not critical for this technique. At this step, test cases are linked to system design documents. Then, total number of function points is calculated by the help of IFPUG method [40]. NTC will be calculated with the Caper's estimation [33].

Product focused testing effort calculation for the products of SP1, SP2 and SP5 can be found on Appendix C.

Comparison between infrastructure based and product focused testing techniques

Cumulative testing cost comparison between two techniques is seen on Figure 4.5. From this figure, testing cost of infrastructure based testing strategy is higher at the start up and then, increasing rate of total testing effort decreases. On the other hand, testing cost of product focused approach is thought as product specific. There is no future investment. Testing cost strongly depends on the function points of the product under test.

These testing strategies can be thought for two different companies. If a company has a role to produce tens or hundreds of software products, it is valuable to use infrastructure based testing. On the other hand, it is not a preferable technique for the company which has an objective to develop a few number of software products. Indeed, smaller company has to use product focused testing strategy.

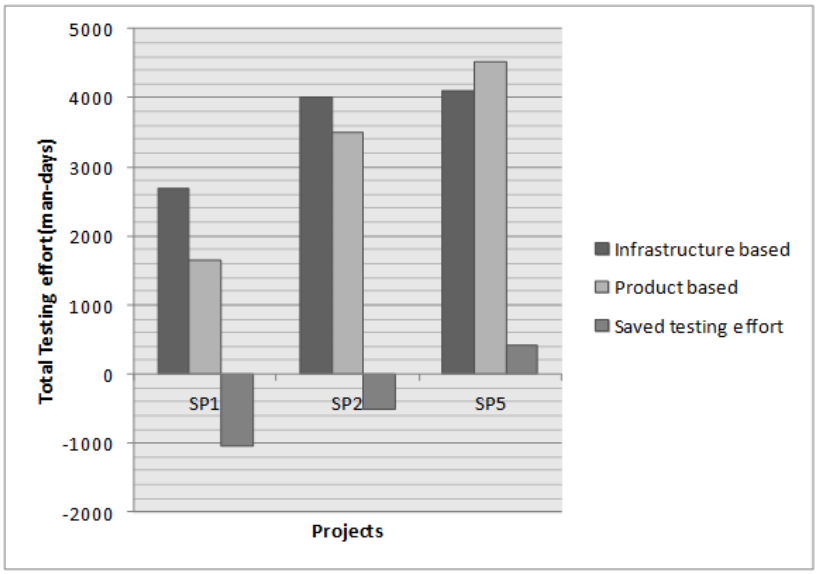


Figure 4.5 - Infrastructure vs. product based testing effort calculation

4.3 Reusability on system tests with testing effort

System test is the final stage of verification. Acceptance test starts after system test. This testing process is applied with the customer before the delivery. It is also named as validation. Indeed, last modifications on software product are done according to system test results and found defects. Therefore, system test has a critical role on the software development process.

Creation of test cases is the most time consuming activity for a test engineer. Although, time is limited for system tests, most of the time is spent to write test cases instead of testing. Reusability methods are offered for the creation of testing artifacts at part 2.4. Requirements based approach for the creation of testing artifacts deals with the reusability of requirements. On the other hand, feature based approach deals with reusability of features.

At Aselsan, there are several products which are developed using SPL technique. Thus, each product has DE process. Software product's system requirements specification document is generated using DE's system requirements and product specific requirements. The process is seen on Figure 4.6.

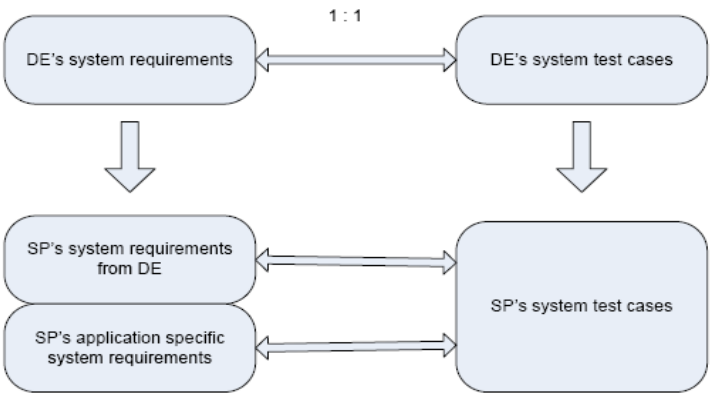


Figure 4.6 - AE system requirements specification document

Test cases are written by referencing DE system requirements' specification document. A requirement specification document is written for the whole domain requirements of TADES during DE analysis.

When DE system requirement's specification document is analyzed, it contains 827 requirements. These requirements are linked to each software products. There are 11 software products which are considered to be delivered to the customer. Number of requirements, which are linked to each software product, can be seen on Table 4.7.

Table 4.7 - Software products' requirements from DE and AE [35]

Products	**No. Of Req. From DE**	**No. Of Product Spec. Req.**	**Total No of Req.**
SP6	233	42	275
SP7	281	55	336
SP8	260	35	295
SP9	110	25	135
SP10	122	24	146
SP11	127	32	159
SP4	454	56	510
SP3	520	45	565
SP12	275	41	316
SP13	257	36	293
SP14	82	21	103
Total	827		

It is restricted that one to one traceability has to be used for the requirement based testing artifact creation. One test case is written for each requirement. When a new system requirement specification document is generated, system test descriptions (STD) are automatically generated with the traceability between requirements and test cases. Then, test cases written for product specific requirements are added to STD document. In brief, after one to one traceability is applied between DE requirements and DE test cases, there will be reusable 827 test cases in total. These test cases can be used for the generation of each product's STD document. Indeed, STD document will be automatically generated by the help of reusability on DE test cases.

There is another method for the creation of testing artifacts. It is the feature tree usage. Feature tree of the domain is implemented during DE. Then, all of the requirements obtained from DE are linked to this feature tree. When a new software product is developed, its feature tree is firstly generated. In fact, SRS document is automatically generated with the features linked to the requirements. The process can be seen on Figure 4.7.

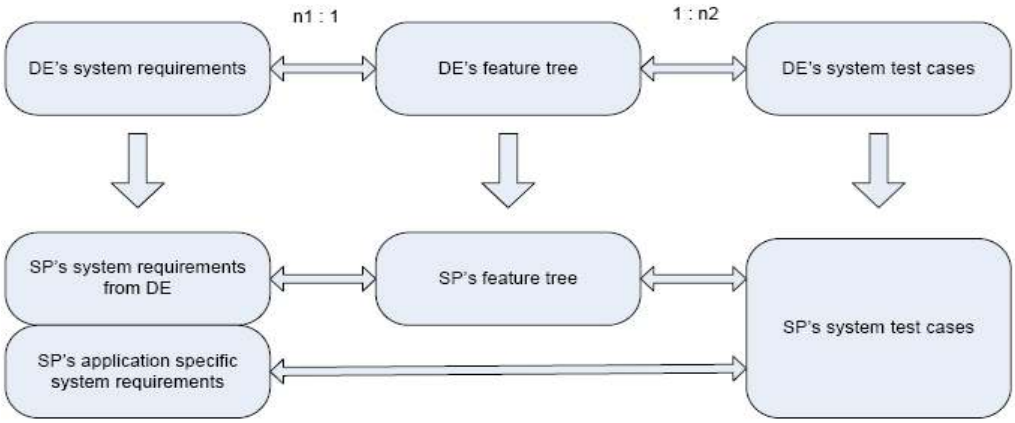


Figure 4.7 - FB traceability between requirements and test cases

Requirements are linked to DE's feature tree with n1 to 1 traceability. Then test cases are linked to DE's feature tree with 1 to n2 traceability. Total NTC can approximately be calculated for the configured feature tree with the following formula.

$$Numberoftestcases = Numberofrequirements \times \frac{1}{n1} \times \frac{n2}{1}$$

If n1 is equal to n2, one to one traceability between requirements and test cases is obtained. Then total NTC is equal to the total number of requirements. NTC can be decreased by feature tree based testing artifacts creation. Increase on the value of n1/n2 results decrease on DE's NTC.

When a comparison is done between requirements and feature tree based testing artifacts creation of Aselsan, NTC for each product can be calculated for DE's system requirements' verification. In addition, testing effort is calculated at the previous study with the help of NTC. If n1 is equal to n2, then model based testing generates the same NTC with the requirements based testing. The value of n1/n2, which is the test model's constraint, is the cost effectiveness indicator of the feature tree based testing artifacts creation. The relationship between the value of n1/n2 and NTC is seen on Figure 4.8.

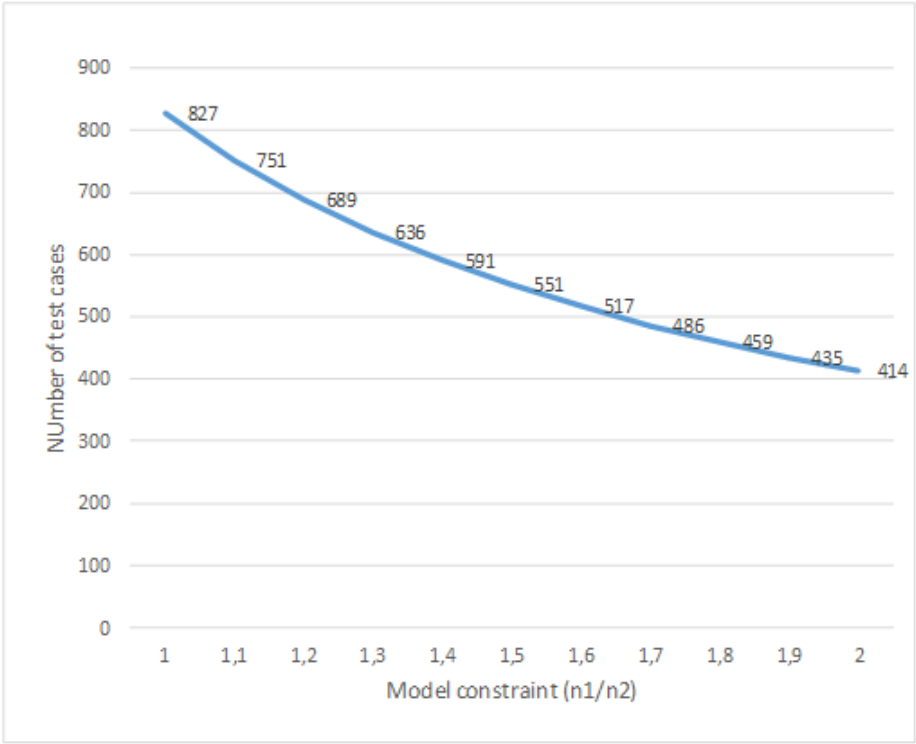


Figure 4.8 - NTC vs. Model constraint

If the value of the model constraint, which is the ratio between the number of requirements and test cases (n1/n2), increases, total NTC decrease. There is an inverse correlation between the value of n1/n2 and total NTC. Indeed, we use the total NTC for each software products' testing effort estimation. If the value of NTC is multiplied with TTP, the result gives testing effort in man hours [41]. As a result, testing effort decreases when the value of model constraint (n1/n2) increases.

4.4 Platform tests at DE

Hundreds of possible configurations can be obtained from the common software components during DE. However, it is impossible to test the whole configurations on DE with system tests because timing is critical for a company. Test managers have to choose a possible way to test the whole configurations to prevent overhead. Redundancy values calculation is offered in part 2.2. In fact, it is used to determine the resembling products.

Redundancy between software products gives idea about the coverage of software products. Software product is verified with platform tests during DE. Found defects on a product and redundancy value calculated with other software products would give a prediction about the defects on other software products.

There is no feature based software development in DE process of Aselsan. Feature tree is not linked to software products. However, common requirements are written at DE level and then they are linked to a software product. Common requirements are used to calculate the redundancy values between software products. Linked requirements to each software product at DE level are seen on Table 4.7.

Redundancy metric is used to give idea about platform test setups of software products. If the redundancy ratio is higher between two software products, they resemble each other. If it is one,

software products are the same. It is not easy to cover the possible combinations at DE level because there may be hundreds of combinations. If it is planned to cover the combinations at DE system level testing, redundancy matrix can be used to eliminate some configurations to reduce the testing effort.

Requirements coverage is the main objective of this study. Since we decrease total number of required test setups, we have to measure requirements coverage of software products. The article gives a mathematical formula for the relationship between TC and TE [36]. Measurement of TE is not directly possible, but it can be calculated with the formula presented in part 2.2.

Redundancy calculation starts by defining same requirements for each product. It is seen on Table 4.8. Then redundancy values between two products are calculated by the help of Table 4.7. Redundancy metric between software products can be seen on Table 4.9.

Table 4.8 - Number of common requirements between SPs [35]

	SP6	SP7	SP8	SP9	SP10	SP11	SP4	SP3	SP12	SP13	SP14
SP6	233	174	168	93	97	101	109	124	79	96	66
SP7	174	281	254	89	110	88	146	165	79	88	80
SP8	168	254	260	90	108	87	147	166	80	89	80
SP9	93	89	90	110	96	106	78	76	56	58	58
SP10	97	110	108	96	122	99	83	88	53	66	67
SP11	101	88	87	106	99	127	74	72	65	65	55
SP4	109	146	147	78	83	74	454	434	176	179	72
SP3	124	165	166	76	88	72	434	520	177	188	77
SP12	79	79	80	56	53	65	176	177	275	118	49
SP13	96	88	89	58	66	65	179	188	118	257	63
SP14	66	80	80	58	67	55	72	77	49	63	82

Table 4.9 - Redundancy values between SPs

	SP6	SP7	SP8	SP9	SP10	SP11	SP4	SP3	SP12	SP13	SP14
SP6	1	0,51	0,51	0,37	0,38	0,38	0,19	0,2	0,18	0,24	0,27
SP7	0,51	1	0,88	0,29	0,38	0,28	0,25	0,26	0,17	0,2	0,28
SP8	0,51	0,88	1	0,32	0,39	0,29	0,26	0,27	0,18	0,21	0,31
SP9	0,37	0,29	0,32	1	0,7	0,81	0,16	0,18	0,17	0,19	0,43
SP10	0,38	0,38	0,39	0,7	1	0,66	0,17	0,16	0,15	0,21	0,49
SP11	0,38	0,28	0,29	0,81	0,66	1	0,15	0,13	0,19	0,2	0,38
SP4	0,19	0,25	0,26	0,16	0,17	0,15	1	0,8	0,32	0,37	0,16
SP3	0,2	0,26	0,27	0,18	0,16	0,13	0,8	1	0,27	0,32	0,17
SP12	0,18	0,17	0,18	0,17	0,15	0,19	0,32	0,27	1	0,29	0,16
SP13	0,24	0,2	0,21	0,19	0,21	0,2	0,37	0,32	0,29	1	0,23
SP14	0,27	0,28	0,31	0,43	0,49	0,38	0,16	0,17	0,16	0,23	1

Resembling software products are chosen from Table 4.9. A higher redundancy value indicates that these software products resemble to each other. If a defect is found on a configuration, it is possible to find the same defect during the testing of other resembling software product because they use same software components. As a result, testing one of these configurations instead of two or many would eliminate the overhead for test engineer.

Software products are divided into groups by using the results obtained from Table 4.9. Higher redundancy value is used for these groupings. Groups of software products can be seen from Table 4.10.

Table 4.10 - Software product DE test groupings

Group Name	Software Products
Group 1	SP6
Group 2	SP7 SP8
Group 3	SP9 SP10 SP11
Group 4	SP4 SP3
Group 5	SP12
Group 6	SP13
Group 7	SP14

One software product will be selected on each groups of 2, 3 and 4 for the verification. If the redundancy values between software products cannot be calculated, 11 different test setups have to be prepared at DE for platform tests. However, there will be only 7 test setups after these groupings of software products. Testing effort will be minimized using this technique.

At this step, requirements coverage of software products will be calculated. It is the requirements coverage metric which can be measured as in part 2.2. Table 4.11 gives the obtained results from the requirements coverage formula. A software product, which has higher number requirements, will be chosen for each group. Then, redundancy values between the reference project and other projects in the group will be used to calculate DE's requirements coverage. The formula will be as in the following.

DE Req. Coverage = (No. of Req. From DE / Total No of Req.) * Redundancy

Table 4.11 – Requirements coverage calculation

Products	No. of Req. From DE	Total No of Req.	Related Test Group	Is it reference product?	Req. coverage w redundancy
SP6	233	275	Group 1	x	0.84
SP7	281	336	Group 2	x	0.84
SP8	260	295	Group 2		0.77
SP9	110	135	Group 3		0.66
SP10	122	146	Group 3		0.55
SP11	127	159	Group 3	x	0.80
SP4	454	510	Group 4		0.71
SP3	520	565	Group 4	x	0.92
SP12	275	316	Group 5	x	0.87
SP13	257	293	Group 6	x	0.87
SP14	82	103	Group 7	x	0.79

TC metrics for the requirements are specified on Table 4.11. TE results comparison would be done with redundancy value usage. At this step, experimental results defined on the article [36] will be used. Probability Density Function (PDF) of coverage is defined as a Gaussian PDF. Then mean

value (μ) can be thought as the calculated coverage values. Standard deviation (σ) is used as 0.1 for each PDF. Then, an example of Gaussian PDF for TC of 0.2 can be found on Figure 4.9.

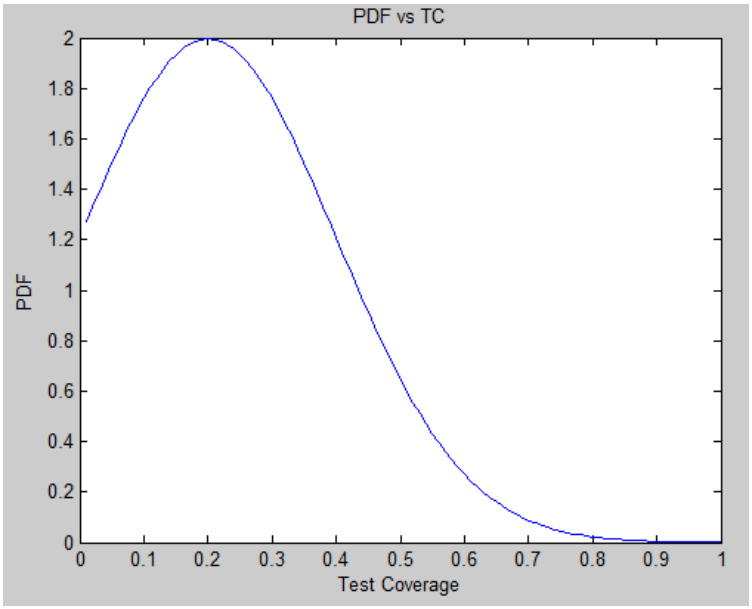


Figure 4.9 – PDF of TC

Then, TE values are calculated by the help of the formula defined at part 2.2. TE values calculated for different TC values can be seen on Figure 4.10.

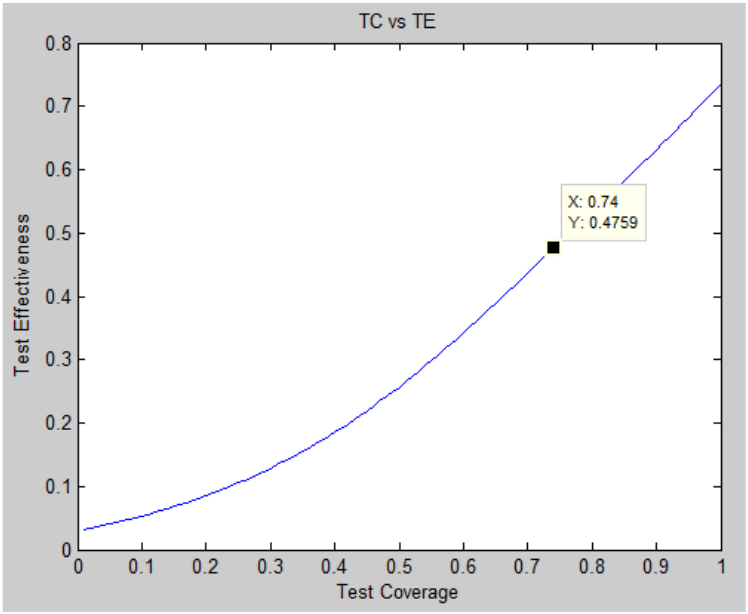


Figure 4.10 – TE vs TC

TE values taken from Figure 4.10 for each software product can be seen on Table 4.12.

Table 4.12 – TE values for platform tests

Products	TE values
SP6	%57
SP7	%57
SP8	%50
SP9	%40
SP10	%30
SP11	%53
SP4	%45
SP3	%65
SP12	%60
SP13	%60
SP14	%52

As a result, testing a reference software project's DE requirements during DE platform tests prevents overhead on the found defects. Although, bolded software products' DE requirements are not verified, they have a TE value using this test setup. It happens because these software products have the same software architecture and they contain same components and defects.

CHAPTER 5

CONCLUSION

This study has focused on Aselsan's SPL testing process from the perspective of testing effort and effectiveness. Four different case studies were carried out focusing on Aselsan's SPL projects, as reported in Chapter 4. These studies were based on the methodologies obtained from the literature. The first study was on testing variation points during DE. It was a study on the creation of test cases differently from the traditional requirement based test case creation method. Possible number of variation point combinations can be seen by the help of feature tree. The proposed technique was compared with the traditional technique from the viewpoint of TE. It was seen that execution of FIG Basis Path method increases TE. The results were shown in Table 4.5. In addition, the proposed technique provides early verification of variation points before variability binding.

The next study was on testing effort comparison between two different testing approaches. Testing variation points before integration of software components prevents increasing number of variation points' combinations. Then, FIG Basis Path method was used to improve effectiveness in the first case study. However, it was necessary to ascertain whether infrastructure based testing is much more effective in terms of testing effort or not. We compared product focused and infrastructure based testing strategies in terms of testing effort. Firstly, FPs were calculated with IFPUG method. Then, they were transformed to NTC via Caper's equation [33]. Testing effort comparison of the two techniques can be seen on Figure 4.5. In summary, the results of this study would lead to a recommendation for companies applying SPL development technique. If a company has an objective to develop many products using SPL, test managers have to choose infrastructure based testing approach. In the future, testing costs will decrease. However, if the company has an objective to produce three or four products, infrastructure based testing is not an effective way of testing. In fact, test manager has to choose product focused testing. In short, this study helps test manager to decide on the selection of a suitable testing method according to the company's objectives.

The third study was on SPL projects' system test case creation method. Requirements based test case creation method is used at Aselsan. There are four methods offered for the creation of test cases in the literature [16, 18, 20, 21]. We have compared requirements based and feature tree based test case creation methods in terms of the total NTC. It is necessary that a test case is written for only one requirement due to traceability in requirement based testing artifacts creation. On the other hand, there is no restriction to write test cases for only one requirement in feature tree based testing artifacts creation. There is a model constraint specified for the calculation of NTC. If the model constraint value increases, NTC decreases. This is seen on Figure 4.8. In summary, if feature tree based reusability is selected and model constraint is set at a higher value, NTC would decrease.

The last study was on platform tests that are performed in the course of DE. Due to the early validation criterion, platform testing is a necessary phase of testing for SPL projects. In fact, it is integrating the common components and then testing them for early validation. Although there are not any platform tests defined at Aselsan, a methodology has been proposed during this study by the help of the calculation of redundancy values between software products. Redundancy values are calculated between 11 possible software products. These values are seen on Table 4.9. Software products were grouped and then a reference project was selected in each group. This reference project was used for platform tests in the course of DE. TC values for each possible software product were calculated. Then, TC metric was used on the offered formula to calculate TE [36]. Obtained results gave us TE values of each software product in platform tests. In short, effort required for platform tests can be decreased.

All studies carried out within the scope of this study were on testing techniques, TC and test management for SPL testing. They were mostly related to TE and testing effort. Testing efficiency is not strictly considered in these studies. In the second case study, which is a comparison between testing techniques in terms of testing effort, testing efficiency metric on two different methods is not considered. In fact, testing efficiency also provides decision support to test manager for the selection of proper testing technique. As a future work, testing efficiency comparison between infrastructure based and product focused testing strategies can be carried out.

Another weakness of this study is related to unit tests. For the verification phases carried out according to V model, unit tests are also a crucial part of verification. Automatic unit test case creation based on SPL architecture may be a future work for SPL testing because unit testing helps to reduce found defects that are related to integration, system or acceptance tests. Tool support for automatic unit test case generation is also necessary for software developers. There are research topics on component analyzing to generate test scripts automatically [25]. In fact, there are some tools for test automation. These tools analyze the software architecture and then they generate test scripts automatically [22]. As an objective to save the software developers' time, these tools can be evaluated for unit testing.

Kauppinen et al [27] have defined hook and template coverage of framework based SPL development. It simply gives an idea about the unit testing methodology. Since TADES is also a framework based development, this study can be applied on TADES as a future work. Increased TE results can be obtained for unit tests.

Another issue that was disregarded in this thesis is test automation. Regression testing decision support techniques were considered. However, a company has to use test automation techniques to decrease testing effort during regression tests, unit tests, integration tests e.g. There are studies offered for test automation for the verification stage of SPL unit testing [22]. Manual testing requires a lot of effort. Therefore, test automation for SPL testing has to be handled as a future work.

REFERENCES

[1] Czarnecki, K.; Generative Programming: “Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models”, Ph. D. thesis, Technische Universität Ilmenau, Germany, 1998, pp. 31-117.

[2] Pressman, R. S.; “Software engineering: a practitioner's approach (sixth edition)”, 2005, pp. 1-64, 495-502.

[3] Pohl, K.; Metzger, A.; “SPL Testing - Exploring principles and potential solutions” in Communications of the ACM, December 2006/Vol.49 No.12, 2006.

[4] Tokcan, M. D.; “Test Effectiveness In An Agile Process-based SPL Environment”, Technical Report METU/II-TR-2009, June 2009.

[5] McGregor, J.; “Testing a Software Product Line”, Technical Report CMU/SEI-2001-TR-022, December 2001.

[6] Lamancha, B.P., Usaola, M.P., Velthius, M.P.; “Software Product Line Testing - A Systematic Review”, Technical Report UCSOFT, 2010.

[7] Pohl, K.; B¨ockle, G.; “Sosftware Product Line Engineering”, Springer, Berlin, 2005, pp. 257-284, 355-370.

[8] IEEE Computer Society, “IEEE Standard for Software Test Documentation”, Technical Report IEEE STD 829-1998, 1998.

[9] Engström, E.; Runeson, P.; “software product line testing - a systematic mapping study”, in Information and Software Technology, 2010.

[10] McGregor, J.; “Structuring Test Assets in a Product Line Effort”, in Workshop #3 at 23rd International Conference on Software Engineering (ICSE2001), Toronto, Ontario, Canada, May 13 ,2001, pp. 89-92.

[11] Jin-hua, L.; Qiong, L.; Jing, Li.; “The W-Model for Testing Software Product Lines”, in 2008 International Symposium on Computer Science and Computational Technology, 2008.

[12] Zeng, H.; Zhang, W.; Rine, D.; “Analysis of Testing Effort by Using Core Assets in Software Product Line Testing”, in International Workshop on SPL Testing August 31, 2004 Boston, Massachusetts, USA, 2004, pp. 1-6.

[13] Buckle, G.; “Calculating ROI for SPLs”, in Vol.21, No:3, IEEE Software, May/June 2004.

[14] Gulechha, L.; “Software Testing Metrics”, available at: http://test.techwell.com/articles/membersub/white-paper-software-testing-metrics , 2009.

[15] Institute of Electrical and Electronics Engineers, “Standard Glossary of Software Engineering Terminology,” in Standard IEEE 610, 12-1990, 1990.

[16] McGregor, J.; “Reasoning about the Testability of Product Line Components” in Co-located with SPLC 2005 - 9th International SPL Conference, 2005, pp. 1-7.

[17] Kolb, R.; Muthig, D.; “Making Testing Product Lines More Efficient by Improving the Testability of Product Line Architectures”, in Workshop on Role of Software Architecture for Testing and Analysis, 2007, pp 22-27. ACM.

[18] Nebut, C.; Fleurey, F.; Traon, Y.; Jézéquel, J.; “A Requirement-based Approach to Test Product Families”, in Proc. Fifth Workshop Product Families Eng., 2009.

[19] Hartmann, J.; Vieira, M.; Ruder, A.; “A UML-based Approach for Validating Product Lines”, in Workshop on Software Product Line Testing (SPLiT), August 2004.

[20] Reuys, A.; Kamsties, E.; Pohl, K.; “Model-Based System Testing of Software Product Families”, in Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE’05), 2005, pp.519-534.

[21] Perrouin, G.; Sen, S.; Klein, J.; Baudry, B.; Traon, Y.; “Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines” in 2010 Third International Conference on Software Testing, Verification and Validation, 2010.

[22] Kauppinen, R.; Taina, J.; “RITA Environment for Testing Framework-based Software Product Lines”, in Proceedings of the Eight Symposium on Programming Languages and Software Tools, June 2003, pp. 58-69.

[23] Sneed, H., M.; “Measuring the Effectiveness of a Test”, available at: http://www.mathematik.uni-ulm.de/sai/mayer/soqua04/slides/sneed.pdf , 2004.

[24] Tahat, L.; Vaysburg, B.; Korel, B.; Bader, A.; “Requirement-Based Automated Black-Box Test Generation”, in Proc. 25th Ann. Int’l Computer Software and Applications Conf., 2001.

[25] Feng, Y.; Liu, X; Kerridge J.; “A product line based aspect-oriented generative unit testing approach to building quality components”, in Proceedings of the IEEE International Workshop Quality-Oriented Reuse of Software (IEEE QUORS’07), 2007.

[26] Olimpiew, E.; Gomaa, H.; “Reusable System Tests for Applications Derived from Software Product Lines”, in International Workshop on Software Product Line Testing (SPLiT 2005), 2005.

[27] Kauppinen, R.; Taina, J.; Tevanlinna, A., “Hook and Template Coverage Criteria for Testing Framework-based Software Product Families”, in Proceedings of the International Workshop on Software Product Line Testing, 2004.

[28] Ganesan, D.; Knodel, J.; Kolb, R.; Haury, U.; Meier, G.; “Comparing costs and benefits of different test strategies for a SPL: A study from testo ag” in Proceedings of the 11th International Software Product Line Conference. Washington, DC, USA: IEEE Computer Society, 2007, pp. 74–83.

[29] Jaring, M.; Krikhaar, R., L.; Bosch, J.; “Modeling Variability and Testability Interaction in SPL Engineering” in Seventh International Conference on Composition-Based Software Systems, 2008.

[30] Cabral, I.; Cohen, M., B.; Rothermel, G.; “Improving the Testing and Testability of Software Product Lines”, in SPLC’10: Proceedings of the 14th International on Software Product Line Conference, 2010.

[31] Engstr¨om, E.; Runeson, P.; “Decision Support for Test Management and Scope Selection in a SPL Context” in 2011 Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011.

[32] Nageswaran, S.; “Test Effort Estimation Using Use Case Points” in Quality Week 2001, San Francisco, California, USA, June 2001.

[33] Capers, J.; “Applied software measurement”, in McGraw-Hill, New York USA, 2nd ed., 1996.

[34] Aselsan Inc., “Teknik Ateş Destek Sistemleri yazılım ürün hattı referans mimari dokümanı”, 2011.

[35] Aselsan Inc., “Teknik Ateş Destek Sistemleri sistem gereksinim özellikleri dokümanı”, 2011.

[36] Ye, R.; Malaiya, Y., K.; “Relationship Between Test Effectiveness and Coverage” in Chillarege Press, Colorado USA, 2002.

[37] Kamsties, E.; Pohl, K.; Reis, S.; Reuys, A.; “Testing Variabilities in Use Case Models” in van der Linden, F. (Ed.): Software Product-Family Engineering – 5th International Workshop (Siena, Italy) , November 2003

[38] What is Test Efficiency and Test Effectiveness? Available at: http://www.softwaretestingdiary.com/2012/05/what-is-test-efficiency-and-test.html

[39] Staats, M.; Whalen, M., W.; Heimdahl, M., P., E.; “Coverage Metrics for Requirements Based Testing: Evaluation of Effectiveness”, in Proceedings of the second NASA Formal Methods Symposium, Washington D.C., USA. , 2010

[40] International Function Point Users Group (IFPUG), “Function Point Counting Practices Manual”, Technical Report ISO/IEC 20926:2003 Release 4.1.1, Michigan, 2002

[41] Sharma, A.; Kushwaha, D., S.; “Applying Requirement Based Complexity For The Estimation Of Software Development And Testing Effort”, in Computer Science and Information Technology, 2012

[42] Zhu, X.; Zhou, B.; Wang, F.; Chen, L.; “Estimate Test Execution Effort at an Early Stage: An Empirical Study”, in International Conference on Cyber World, IEEE Computer Society, 2008, pp-195-200

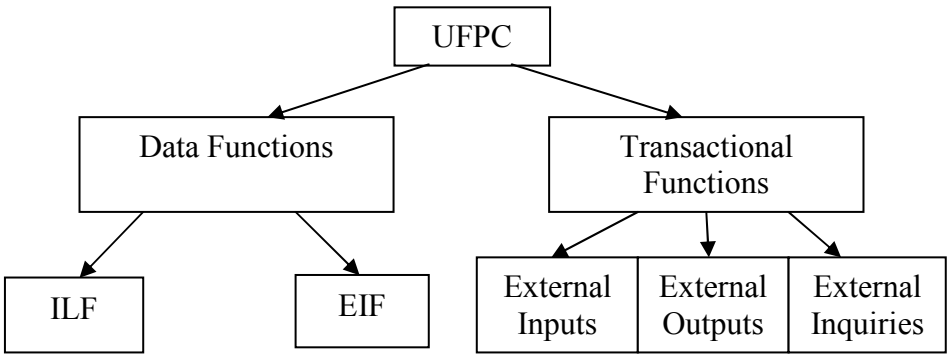
APPENDIX A

TESTING EFFORT CALCULATION USING IFPUG METHOD

Function point is the leading metric for the software development industry. They are mostly used for development and testing effort estimations. It is measuring software size from the user's point of view. As the usage of function point metric increases through companies, function point calculation software programs are used in the industry. At this step, IFPUG function point calculation methodology will be issued [40].

Firstly, Unadjusted Function Point Count (UFPC), which is countable measurement from the view point of user, will be issued. UFPC is divided into two categories. They are data functions and transactional functions. The categories for UFPC can be seen on Figure A.1.

Figure A.1- UFPC Measurement



An internal logical file (ILF) is a logical data or control information which is identified by user among the boundary of the application. The primary intent of an ILF is to contain data that are preserved through one or more elementary processes of the application.

An external interface file (EIF) is a logical data or control information which is identified by user among the boundary of another application. The primary purpose of an EIF is to contain data that are preserved through one or more elementary processes of the application.

An external input (EI) is an elementary process which handles data or control information coming from the external interfaces of the software product. EI contains one or more ILFs or, it is used to change the system's behavior.

An external output (EO) is an elementary process which sends data or control information to the outside boundary of software application. EO contains one or more ILFs or, it is used to change the system's behavior.

An external inquiry (EQ) is an elementary process which sends data or control information to the outside boundary of software application. EQ contains no ILFs or, it is not used to change the system's behavior.

Type of IFPUG projects

IFPUG function points are measured for three types of projects. These projects are development projects, enhancement projects and application. Function points calculated for development projects

are the measures of functions from the point of view of user at the first delivery of the project. Function points calculated for enhancement projects are the measures of functions when the product is updated. Then function points calculated for the application is the last measure of function points on the installed product.

UFPC Calculation for Data Functions

Record Element Type (RET) is the user identified groups for ILFs and EIFs. RETs can be divided into two categories which are mandatory and optional subgroups. Data Element Type (DET) is a non-repeatable field that is recognized by user. Functional complexity will be measured by the help of Table A.1.

Table A.1 – Functional Complexity Matrix

	1 to 19 DET	20 to 50 DET	51 or more DET
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

Then, functional complexity value will be used for the selection unadjusted function points (UFP) values for ILF. The selection will be done using Table A.2.

Table A.2 – UFP values for ILF

Functional Complexity Rating	UFP
Low	7
Average	10
High	15

UFP values of EIF will also be selected by the help of functional complexity matrix. The selection will be done according to Table A.3.

Table A.3 – UFP values for EIF

Functional Complexity Rating	UFP
Low	5
Average	7
High	10

UFPC Calculation for Transactional Functions

File Type Reference (FTR) is an ILF that is internal or external file which is read or provided by a transactional function. Data Element Type (DET) is a non-repeatable field that is recognized by user. Functional complexity for EIs will be measured by the help of Table A.4.

Table A.4 – Functional Complexity Matrix

	1 to 19 DET	20 to 50 DET	51 or more DET
0 to 1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3 or more FTRs	Average	High	High

Functional complexity for EOs or EQs will be measured by the help of Table A.5.

Table A.5 – Functional Complexity Matrix

	1 to 19 DET	20 to 50 DET	51 or more DET
0 to 1 FTR	Low	Low	Average
2 to 3 FTRs	Low	Average	High
4 or more FTRs	Average	High	High

Then, functional complexity value will be used for the selection of UFP values for EIs and EQs. The selection will be done using Table A.6.

Table A.6 – UFP values for ILF

Functional Complexity Rating	UFP
Low	3
Average	4
High	6

UFP values of EOs will also be selected by the help of functional complexity matrix. The selection will be done according to Table A.7.

Table A.7 – UFP values for EIF

Functional Complexity Rating	UFP
Low	4
Average	5
High	7

Test environment complexity factor (TEF) depends on different variables. It is preferable that test manager has an experience for the testing process to have an idea about the values of the variables. He specify a weight for each factor and then multiply the assigned values with the corresponding weight and then the whole calculated values for factors are summed up. Table A.8 gives the factor dependencies of TEF value.

Table A.8 - TEF calculation values [32]

Factor	**Description**	**Assigned value**
T1	Test tools	5
T2	Documented inputs	5
T3	Development Environment	2
T4	Test Environment	3
T5	Test-ware reuse	3
T6	Distributed system	4
T7	Performance objectives	2
T8	Security Features	4
T9	Complex interfaces	5

After the calculation of TEF value, the formula defined for Adjusted Function Points (AFP) is given as:

AFP= UFP *[0.65+ (0.01*TEF)]

AFP is then used for the calculation of NTC with Caper's estimation formula [33].

NTC = AFP ^ 1.2

Then, the last step for the estimation of testing effort is multiplying NTC with TTP metric. Testing effort will be measured with the following formula [41].

Testing Effort = NTC x TTP (man-hours)

APPENDIX B

INFRASTRUCTURE BASED TESTING EFFORT CALCULATION

If infrastructure based testing strategy is applied, then testing effort will be examined with reusable, adaptation and product specific types. Number of linked requirements to these costs can be seen on Table B.1.

Table B.1 - Number of Requirements for SP1

Software Component	*Creusable*	*Cadapt*	*Cproduct _ specific*
Software Component 0 (SC0)	124	-	-
Software Component 2 (SC2)	141	-	-
Software Component 6 (SC6)	40	-	-
Software Component 7 (SC7)	34	-	-
Software Component 8 (SC8)	64	-	-
Software Component 9 (SC9)	27	-	-
Software Component 10 (SC10)	122	-	-
Software Component 11 (SC11)	41	-	-
Software Component 12 (SC12)	47	-	-
Software Component 13 (SC13)	56	-	-
Software Component 14 (SC14)	78	-	-
Software Component 15 (SC15)	49	-	-
Software Component 16 (SC16)	42	-	-
Software Component 17 (SC17)	29	-	-
Software Component 18 (SC18)	222	-	-
Software Component 19 (SC19)	36	-	-
Software Component 20 (SC20)	-	-	144
Software Component 21 (SC21)	-	-	105

Number of UFPs for SCs is calculated by the help of IFPUG analysis on the requirements. Number of UFPs can be seen on Table B.2.

Table B.2 – UFPs for SP1

Software Component	*Creusable*	*Cadapt*	*Cproduct_specific*
Software Component 0 (SC0)	26	-	-
Software Component 2 (SC2)	34	-	-
Software Component 6 (SC6)	22	-	-
Software Component 7 (SC7)	14	-	-
Software Component 8 (SC8)	25	-	-
Software Component 9 (SC9)	10	-	-
Software Component 10 (SC10)	30	-	-
Software Component 11 (SC11)	21	-	-
Software Component 12 (SC12)	18	-	-
Software Component 13 (SC13)	29	-	-
Software Component 14 (SC14)	32	-	-
Software Component 15 (SC15)	22	-	-
Software Component 16 (SC16)	20	-	-
Software Component 17 (SC17)	17	-	-
Software Component 18 (SC18)	95	-	-
Software Component 19 (SC19)	18	-	-
Software Component 20 (SC20)	-	-	102
Software Component 21 (SC21)	-	-	50

Number of UFPs is calculated as 585. At this step, TEF value will be calculated.

Table B.3 - TEF Calculation for SP1

Factor	**Assigned Value**	**Weight**	**Extended Value**
T1	5	1	5
T2	5	2	10
T3	2	5	10
T4	3	4	12
T5	3	3	9
T6	4	4	16
T7	2	1	2
T8	4	2	8
T9	5	4	20
Total			92

AFP =UFP *[0.65+ (0.01*TEF)] = 585 *[0.65+ (0.01*92)] =918.45

NTC = AFP ^ 1.2 = 3594.7
TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 3594.7 * 6
= 21568.3 (man-hours) = 2696.8 (man-days)

Total testing effort for SP1 is calculated as 2696.8 man-days. Number of requirements linked to software components for the software product of SP2 can be seen on Table B.4.

Table B.4 - Number of Requirements for SP2

Software Component	*Creusable*	*Cadapt*	*Cproduct_specific*
Software Component 0 (SC0)	-	60	-
Software Component 2 (SC2)	-	108	-
Software Component 6 (SC6)	-	24	-
Software Component 7 (SC7)	-	-	-
Software Component 8 (SC8)	-	27	-
Software Component 9 (SC9)	-	41	-
Software Component 10 (SC10)	-	60	-
Software Component 11 (SC11)	-	150	-
Software Component 12 (SC12)	-	3	-
Software Component 13 (SC13)	-	104	-
Software Component 14 (SC14)	-	41	-
Software Component 19 (SC19)	-	-	-
Software Component 22 (SC22)	129	-	-
Software Component 23 (SC23)	94	-	-
Software Component 1 (SC1)	163	-	-
Software Component 24 (SC24)	-	-	398
Software Component 25 (SC25)	-	-	175

Number of UFPs for SCs is calculated by the help of IFPUG analysis on the requirements. Number of UFPs can be seen on Table B.5.

Table B.5 – UFPs for SP2

Software Component	*Creusable*	*Cadapt*	*Cproduct_specific*
Software Component 0 (SC0)	-	13	-
Software Component 2 (SC2)	-	25	-
Software Component 6 (SC6)	-	7	-
Software Component 7 (SC7)	-	-	-
Software Component 8 (SC8)	-	10	-
Software Component 9 (SC9)	-	5	-
Software Component 10 (SC10)	-	17	-
Software Component 11 (SC11)	-	27	-
Software Component 12 (SC12)	-	3	-
Software Component 13 (SC13)	-	15	-
Software Component 14 (SC14)	-	8	-
Software Component 19 (SC19)	-	-	-
Software Component 22 (SC22)	27	-	-
Software Component 23 (SC23)	17	-	-
Software Component 1 (SC1)	33	-	-
Software Component 24 (SC24)	-	-	83
Software Component 25 (SC25)	-	-	33

Number of UFPs is calculated as 323. At this step, TEF value will be calculated.

Table B.6 - TEF Calculation for SP2

Factor	Assigned Value	Weight	Extended Value
T1	5	1	5
T2	5	2	10
T3	2	5	10
T4	3	4	12
T5	3	3	9
T6	4	4	16
T7	2	1	2
T8	4	2	8
T9	5	4	20
Total			92

AFP =UFP *[0.65+ (0.01*TEF)] = 323 *[0.65+ (0.01*92)] =507.11

NTC = AFP ^ 1.2 = 1762.4

TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 1762.4 * 6
= 10574.4 (man-hours) = 1321 (man-days)

Total testing effort for SP2 is calculated as 1321 man-days. Number of requirements linked to software components for the software product of SP5 can be seen on Table B.7.

Table B.7 - Number of Requirements for SP5

Software Component	*Creusable*	*Cadapt*	*Cproduct_specific*
Software Component 0 (SC0)	-	-	-
Software Component 2 (SC2)	-	-	-
Software Component 6 (SC6)	-	-	-
Software Component 7 (SC7)	-	-	-
Software Component 8 (SC8)	-	-	-
Software Component 9 (SC9)	-	-	-
Software Component 10 (SC10)	-	-	-
Software Component 11 (SC11)	-	-	-
Software Component 12 (SC12)	-	-	-
Software Component 13 (SC13)	-	-	-
Software Component 14 (SC14)	-	-	-
Software Component 19 (SC19)	-	-	-
Software Component 22 (SC22)	-	-	-
Software Component 23 (SC23)	-	-	-
Software Component 26 (SC26)	-	-	84
Software Component 27 (SC27)	-	-	73

Number of UFPs for SCs is calculated by the help of IFPUG analysis on the requirements. Number of UFPs can be seen on Table B.8.

Table B.8 – UFPs for SP5

Software Component	*Creusable*	*Cadapt*	*Cproduct_specific*
Software Component 0 (SC0)	-	-	-
Software Component 2 (SC2)	-	-	-
Software Component 6 (SC6)	-	-	-
Software Component 7 (SC7)	-	-	-
Software Component 8 (SC8)	-	-	-
Software Component 9 (SC9)	-	-	-
Software Component 10 (SC10)	-	-	-
Software Component 11 (SC11)	-	-	-
Software Component 12 (SC12)	-	-	-
Software Component 13 (SC13)	-	-	-
Software Component 14 (SC14)	-	-	-
Software Component 19 (SC19)	-	-	-
Software Component 22 (SC22)	-	-	-
Software Component 23 (SC23)	-	-	-
Software Component 26 (SC26)	-	-	13
Software Component 27 (SC27)	-	-	17

Number of UFPs is calculated as 30. At this step, TEF value will be calculated.

Table B.9 - TEF Calculation for SP5

Factor	**Assigned Value**	**Weight**	**Extended Value**
T1	5	1	5
T2	5	2	10
T3	2	5	10
T4	3	4	12
T5	3	3	9
T6	4	4	16
T7	2	1	2
T8	4	2	8
T9	5	4	20
Total			92

AFP =UFP *[0.65+ (0.01*TEF)] = 30 *[0.65+ (0.01*92)] =47.1

NTC = AFP ^ 1.2 = 101.7
TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 101.7 * 6
= 610.2 (man-hours) = 76.3 (man-days)

Total testing effort for SP5 is calculated as 76.3 man-days.

APPENDIX C

PRODUCT FOCUSED TESTING EFFORT CALCULATION

Total number of test requirements for SP1, SP2 and SP5 is seen from Table C.1.

Table C.1 - Number of software products' requirements

Software Product	Total Number of Requirements
SP1	1305
SP2	1424
SP5	1193

Table C.2 - Number of UFPs of software products

Software Product	Number of UFPs
SP1	382
SP2	420
SP5	253

Table C.3 - TEF Calculation

Factor	Assigned Value	Weight	Extended Value
T1	5	1	5
T2	5	3	15
T3	2	1	2
T4	3	2	6
T5	3	5	15
T6	4	5	20
T7	2	1	2
T8	4	5	20
T9	5	2	10
			95

Testing effort calculation for the project SP1 product focused approach

AFP =UFP *[0.65+ (0.01*TEF)] = 382 *[0.65+ (0.01*95)] =611.2

NTC = AFP ^ 1.2 = 2205

TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 2205 * 6
= 13230 (man-hours) = 1653.8 (man-days)

Total testing effort for SP1 is calculated as 1653.8 man-days.

Testing effort calculation for the project SP2 product focused approach

AFP =UFP *[0.65+ (0.01*TEF)] = 420 *[0.65+ (0.01*95)] =672

NTC = AFP ^ 1.2 = 2470.8

TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 2470.8* 6
= 14825 (man-hours) = 1853 (man-days)

Total testing effort for SP2 is calculated as 1853 man-days.

Testing effort calculation for the project SP5 product focused approach

AFP =UFP *[0.65+ (0.01*TEF)] = 253 *[0.65+ (0.01*95)] =405

NTC = AFP ^ 1.2 = 1345

TTP = 6 man-hours are required to plan, write and execute tests , 3 Test engineers with rank of 2 due to less knowledge about the application domain.

Testing Effort = NTC * TTP = 1345* 6
= 8069 (man-hours) = 1009 (man-days)

Total testing effort for SP5 is calculated as 1009 man-days.

APPENDIX D

TEST METRICS

D.1 Defect Metrics [4]

Defects by Injection Phase

This metric is collected on the SPL development phases. It is the separation of the found defects from which stage they are injected. Defects can be related to DE, AE or variability binding. If most of the defects are collected during DE processes, DE development is a problematic stage of the development.

Defects by Detection Phase

This metric helps to specify the most effective testing process of SPL stages. If NOD found on DE are higher than NOD found on AE. Thus, testing is more effective on DE phase.

Defects by Action Taken

Defects collected from SPLE can be categorized on its state. If the percentage of defects that are on the state of 'Not a defect', test engineer does not have a good knowledge on the concept of development.

Defects by Priority

It is the classification of defects depending on their priority. Defects are classified for the priority states of 'High', 'Medium' or 'Low'.

Defects by Cause

This metric helps the development team to determine sub-processes to be improved. Appropriate causes are identified for the specification of the development processes.

Defects by Type

This metric helps the development team to classify of the defects from types. Proper types of defects can be determined.

D.2 Test Coverage Metrics [14]

Test Case Productivity

Test case productivity is defined by the formula in [14] as:

$$Test\ Case\ Productivity\ = \left[\frac{Total\ Raw\ Test\ Steps}{Efforts\ (hours)} \right] Step(s)/hour$$

It is the creation time for the test cases. If it is higher, then time to create testing artifacts will decrease. For SPL testing, by the help of reusability of testing artifacts, the metrics can be obtained and compared with the previous case.

Test Execution Summary

It is the execution summary of the test cases. If the test cases are ready for testing, the software sometimes cannot be ready for the execution of the overall test cases. This is the metric for testing. The test case can be on one of the states.

i. Fail
ii. Pass
iii. Not executed.

Defects Acception

With the test report, defects are directed to the software development team. Some defects can be founded as valid or not valid defects. It is simply related to Learning of the Software Testing Team. Defects acceptation is a metric that defines the valid defects. It is really connected to the Learning Effort of SPL Testing. The equation is defined as :

$$Defect\ Acceptance = \left[\frac{Number\ of\ Valid\ Defects}{Total\ Number\ of\ Defects} * 100 \right] \%$$

Defects Rejection

Defects Rejection is a metric that defines the invalid defects. The equation is defined as: [14]

$$Defect\ Rejection = \left[\frac{Number\ of\ Defect(s)\ Rejected}{Total\ Number\ of\ Defects} * 100 \right] \%$$

Bad Fix Defect

Bad Fix Defect is the metric for the found defects which are the reason for new defects. Bad Fix Defects are the determined defects which are found on the common components of SPL. The equation is defined by [14]

$$Bad\ Fix\ Defect = \left[\frac{Number\ of\ of\ Bad\ Fix\ Defect(s)}{Total\ Number\ of\ Valid\ Defects} * 100 \right] \%$$

Test Execution productivity

It is the metric that shows the test cases execution per day. The equation is defined as:

$$Test\ Execution\ Productivity = \left[\frac{Total\ No.\ of\ TC\ executed\ (Te)}{Execution\ Efforts\ (hours)} * 8 \right] Execution(s)/Day$$

Where Te is calculated as,

$$Te = Base\ Test\ Case + ((T\ (0.33) * 0.33) + (T\ (0.66) * 0.66) + (T\ (1) * 1))$$

Where,
Base Test Case = No. of Test Cases executed atleast once.

T (1) = No. of Test Cases Retested with 71% to 100% of Total TC steps
T (0.66) = No. of Test Cases Retested with 41% to 70% of Total TC steps
T (0.33) = No. of Test Cases Retested with 1% to 40% of Total TC steps

Test Efficiency

It is the metric which shows the results of the applied testing efficiency. The defects found during testing phase and the defects that come from the customer.

$$Test\ Efficiency = \left[\frac{DT}{DT + DU} * 100 \right] \%$$

Where,
DT = Number of valid defects identified during testing.
DU = Number of valid defects identified by user after release of application. In other words, post-testing defect

Defect Severity Index

It is the metric that gives idea about the product's quality to release. If this metric is higher, then the product is not ready to release. It is defined as:

$$Defect\ Severity\ Index = \left[\frac{\sum (Severity\ Index * No.\ of\ Valid\ Defect(s)\ for\ this\ severity}{Total\ Number\ of\ Valid\ Defects} \right]$$

Effort Variance

It gives the variance in the estimated effort. It is defined by the equation:

$$Effort\ Variance = \left[\frac{Actual\ Effort - Estimated\ Effort}{Estimated\ Effort} * 100 \right] \%$$

Schedule Variance

It gives the variance in the estimated schedule, then the equation is:

$$Schedule\,Variance = \left[\frac{Actual\,No.\,of\,Days - Estimated\,No.\,of\,Days}{Estimated\,No.\,of\,Days} * 100 \right] \%$$

Scope Variance

It indicates how the scope of testing is stable.

$$Scope\,Change = \left[\frac{Total\,Scope - Previous\,Scope}{Previous\,Scope} * 100 \right] \%$$

Where,
Total Scope = Previous Scope + New Scope, if Scope increases
Total Scope = Previous Scope - New Scope, if Scope decreases