DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

AN INTEGRATED MEASUREMENT AND EVALUATION METHOD FOR SOFTWARE DEVELOPERS' PERFORMANCE

by
Mustafa BATAR

July, 2015
İZMİR

AN INTEGRATED MEASUREMENT AND EVALUATION METHOD FOR SOFTWARE DEVELOPERS' PERFORMANCE

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of
Science in Computer Engineering

by
Mustafa BATAR

July, 2015
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"AN INTEGRATED MEASUREMENT AND EVALUATION METHOD FOR SOFTWARE DEVELOPERS' PERFORMANCE"** completed by **MUSTAFA BATAR** under supervision of **ASST. PROF. DR. KÖKTEN ULAŞ BİRANT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Kökten Ulaş BİRANT

Supervisor

Prof. Dr. [illegible]

(Jury Member)

Yrd. Doç. Dr. Özgür Yeniay

(Jury Member)

Prof. Dr. Ayşe OKUR
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank all who helped me to make this study. In particular, I want to express gratefulness to the following people:

Firstly, I should thank to the greatest Turkish revolutionist and anti-imperialist leader who is the founder of Republic of Turkey - Ghazi M. Kemal ATATÜRK for having education in the light of science in the universities.

To my parents - Ertuğrul BATAR & Şerife BATAR for their support and for teaching me to see, to read, to learn and to think. If they did not be with me, I could not complete my thesis.

To my thesis supervisor, Asst. Prof. Dr. Kökten Ulaş BİRANT, for his guidance and contributions during my studies and trust in my work. During the process of writing my thesis, he was sometimes a professor of me, sometimes a brother of me, sometimes a friend of me and sometimes a father of me as well as always, he became fair, and treated fairly to me.

To my professor, Asst. Prof. Dr. Derya PAKALIN BİRANT, for her contributions and useful suggestions through this study.

To my favourite singers who I listened to while I was writing my thesis, Nurettin RENÇBER, Cevdet BAĞCA, Onur AKIN, Edip AKBAYRAM, Neşet ERTAŞ, Hüsnü ARKAN, Birsen TEZER, Linet, Özlem ÖZDİL, Zülfü LİVANELİ and Ahmet KAYA.

Lastly and most morally, to my football team, Beşiktaş - also known as Karakartal for teaching me to having a conscience, to live honorably and honestly by this motto: *"Play with Honour, Win Rightly."*

Mustafa BATAR

AN INTEGRATED MEASUREMENT AND EVALUATION METHOD FOR SOFTWARE DEVELOPERS' PERFORMANCE

ABSTRACT

Up to now, several criterions have been determined in order to evaluate software developers' performance: Productivity, Engagement, Attention to Quality, Code Base Knowledge and Management, Adherence to Coding Guidelines and Techniques, Learning and Skills, Personal Responsibility and etc. However, there isn't any universally accepted methodology to measure and evaluate software developers' performance. There are three main reasons of this situation: Firstly, each part of software creation is unique. There is no compelling reason to assemble two times the same parts of software as it might be duplicated by copying it. This makes it truly difficult to make a formal and thorough correlation between two parts of software. Secondly, the current technology is something that changes at a truly fast pace. So, each time a methodology in respect to a certain wave of technology is dependable enough, it is for the most part as of recently old. Thirdly, there is a gigantic zone for innovativeness in discovering the diverse answers for a unique issue.

About the thesis subject, the background has been prepared before the study of thesis by observing and analyzing the researches which have been done before and the case studies which have been published before. With this background study, the common criteria set about the measurement and evaluation of software developers' performance have been presented. Some information has been got from some software developers and managers by doing survey technic on internet so as to evaluate the use of the common criteria in real work life and identify criteria which are used in real work life but not seen in researches before. In the light of the survey results, a measurement and evaluation criteria set about the software developers' performance have been created.

Keywords: Software engineering, software developers, performance.

YAZILIM GELİŞTİRİCİLERİNİN ÇALIŞMA PERFORMANSLARININ ÖLÇÜLMESİ VE DEĞERLENDİRİLMESİNE YÖNELİK BÜTÜNLEŞİK BİR YÖNTEM

ÖZ

Şimdiye kadar, yazılım geliştiricilerinin çalışma performanslarını ölçmek ve değerlendirmek amacıyla çeşitli kıstaslar belirlenmiştir: üretkenlik, taahhüt, kaliteye önem verme, koda dayalı bilgi ve değerlendirme, kodlamanın kılavuz ve kurallarına uygunluk, öğrenme becerisi, kişisel sorumluluk bilinci, vb. Ancak, yazılım geliştiricilerinin çalışma performanslarını ölçmek ve değerlendirmek amacıyla dünyaca kabul görmüş herhangi bir yöntem yoktur. Bu durumun üç temel nedeni vardır: Birinci sebep: geliştirilen her bir yazılım parçası tektir. Fakat, aynı yazılım parçasını geliştirmek için onu yeniden sil baştan yaratmaya gerek yoktur; elimizde var olan yazılım parçasını kopyalayarak bu sorun çözülebilmektedir. Bu da, iki yazılım parçası arasında nitelik ve nicelik bakımından tam doğru bir karşılaştırmanın yapılamamasına neden olmaktadır. İkinci sebep: günümüz teknolojisi sürekli değişen ve gelişen bir süreç içerisindedir. Bunun sonucunda, yazılım geliştiricilerinin çalışma performanslarını ölçen ve değerlendiren bir yöntemin kullandığı oldukça güvenilir bir teknoloji çok geçmeden önemini kaybetmiş ve eskimiş olmaktadır. Böylece, bu yöntem işe yaramaz duruma gelmektedir. Üçüncü sebep: aynı problemi çözmek için birden çok, birbirinden tamamen farklı çeşitli yöntemler geliştirilip yaratıcılık kavramı had safhaya çıkarılabilmektedir. Örneğin, aynı probleme çözüm üretmeye çalışan yazılım geliştiricilerinin yazdığı kodların satır sayılarını hesaplamak, problemin büyüklüğünü değil, çözümün büyüklüğünü ölçmek demektir.

Konu ile ilgili olarak hâli hazırda yapılmış araştırmalar ve yayınlanmış vaka analizleri incelenerek özetlenmiş, çalışma öncesi altyapı hazırlanmıştır. Oluşturulan altyapı ile çalışan değerlendirilmesine yönelik bilinen kıstaslar ortaya konmuştur. Belirlenen kıstasların gerçek hayatta kullanımının değerlendirilmesi ve gerçek hayatta kullanılmakta olan ancak araştırmalarda gözlenmeyen kıstasların tanımlanması adına yazılım çalışanları ve yöneticileri arasında anket ve görüşme

teknikleri yoluyla bilgi toplanmıştır. Ortaya çıkan sonuçlar ışığında, yazılım geliştiricilerinin çalışma performanslarına yönelik olarak bir ölçme ve değerlendirme kıstasları seti oluşturulmuştur.

Anahtar kelimeler: Yazılım mühendisliği, yazılım geliştiricileri, performans.

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 General Information and Purpose

Measurement and evaluation of software developers' performance is a seriously hard task. Software projects often have objectives such as, maintainability, reliability, efficiency, security, flexibility, testability and etc. Unfortunately, it is often unclear which software developer deals with which part of a software project and how their change affects those goals. This expert proposition has tried to empower an evaluator to gain a good overview of how software developers perform in a software project.

In this master thesis, firstly the survey about the paper has been prepared by the contribution of seventeen articles with scientific index which have been published in IEEE or ACM digital libraries and one master thesis which have been completed in 2008. This survey has included sixty three criteria set about personal information, program development, design and management, program completion process and working place atmosphere.

Secondly, it has been determined which companies, where several software developers work, will conduct the survey. More than one hundred companies have been chosen; however, it is known that some software developers who work at fifty five of them have taken the survey.

After that, the survey has been conducted by one hundred and five software developers for about three weeks. Based on the coming answers, we have seen that ten of the participants haven't answered any questions and fourteen of them have answered the questions in the survey illogically. For instance, someone has declared while s/he has written one code line for a day, s/he has written one thousand comment lines for a day while developing a program; or someone has declared while s/he has done one thousand build operations for a day, s/he hasn't developed any

codes for a day while developing a program. In the end, we can say that eighty one of the participants, who have taken the survey, have answered all the questions in the survey logically.

After the clarification operation of the participants, who have answered the questions illogically in the survey, has been done, some statistical operations about the participants, who have answered the questions logically in the survey, have been done in the windows application "excel". As a result of this, the number of each answer of each question in the survey has been graphed, and so visualized in the windows application "excel".

After the graphing operation in the windows application "excel" has been done, in order to see the relationship among each criterion in the criteria set in the survey, two data mining algorithms - Association Rule Mining Apriori Algorithm and Classification C4.5 Algorithm - have been implemented. Based on these algorithms, it has been understood that ten criteria in the criteria set in the survey have been answered in the same direction by the participants (software developers), and also, six bilateral relations among these ten criteria in the criteria set in the survey have been determined.

In conclusion, in the light of the survey results, a measurement and evaluation criteria set about the software developers' performance have been created. Furthermore, this master thesis provides reliable perspective to authorized people about how to measure and evaluate software developers' performance while they are working in a software project.

1.2 Organization of the Thesis

This thesis includes seven chapters and the remaining of this thesis has been organized as follows:

In Chapter 2, the terms “measurement”, “evaluation”, “software measurement” and “software evaluation” have been explained in detail.

In Chapter 3, general information about how to measure and evaluate software developers’ performance has been given.

In Chapter 4, general information about the articles about measurement and evaluation of software developers’ performance, which have been determined after literature review, has been given.

In Chapter 5, general information about the questions in the survey has been given and their answers have been explained in detail.

In Chapter 6, criteria which have been determined based on the survey results and their relationships between each other have been explained in detail.

Finally, in Chapter 7, the conclusion remarks and future works have been given.

CHAPTER TWO
MEASUREMENT AND EVALUATION

2.1 What is Measurement?

Estimation is the task of numerals to questions or occasions as indicated by some standards. Numerals are the marks which have no inalienable significance, such as postal districts and car tags. Numbers are the numerics which have quantitative importance and could be investigated, such as weight and height.

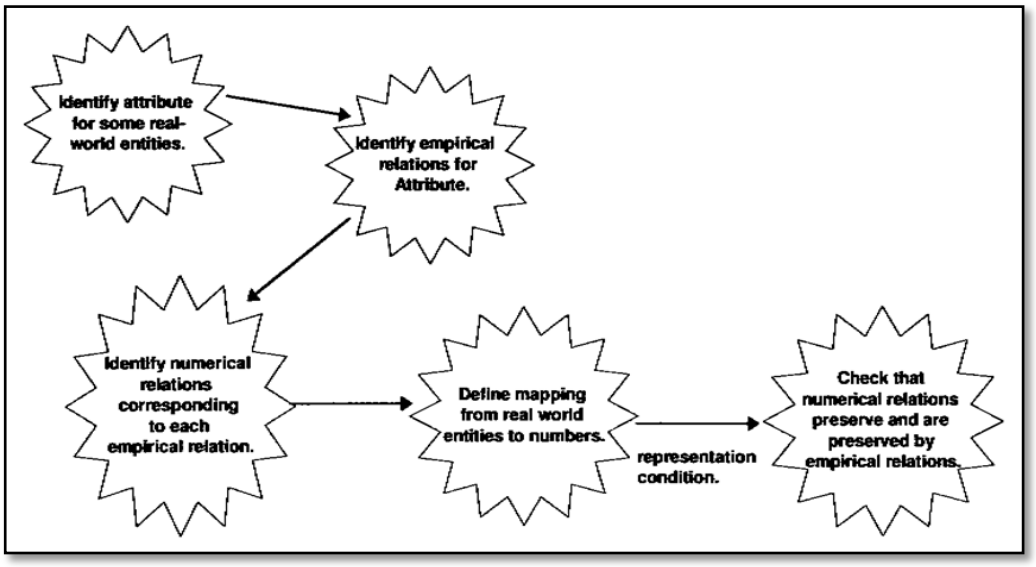

Figure 2.1 Measuring

The guidelines for allotting marks to properties of variables are the most critical segments of estimation, in the light of the fact that the consequence of poor principles is futile results. Ideas frequently could not be estimated straightforwardly, so what are generally estimated are markers of builds; for example, speed, rationale, verbal aptitude and so on (Michael, 2008).

2.1.1 Measurement Types

Four scales which belong to estimation have been distinguished. Those scales vary in how nearly they come close to the construction of the number framework which we utilize. Finding out the scale of estimation of attributes utilized as a part of examination is critical on the grounds that the type of estimation decides the sorts of factual investigations that can be directed. The conclusions that can be drawn from exploration rely on upon the factual examination utility.

2.1.1.1 Type - Nominal

Ostensible type of estimation utilizes images to characterize perceptions into totally unrelated and comprehensive classes. Totally unrelated thing means that the classifications have to be unmistakable so that no perception takes a part more than one class. Comprehensive thing means that sufficient classifications have to exist so that all perceptions take a part in some classification.

That is the most essential type of estimation. At this type, we could focus just whether two perceptions are similar or diverse. A case: In a study of educators, gender was controlled by an inquiry. Perceptions were categorized into two totally unrelated and thorough classes - female and male. Perceptions could be marked with the letters “F” and “M” or the numbers 0 and 1.

At the same overview, the variable of military status could be estimated by two classes - wedded and unwedded. At the same time, these classes should be characterized so that every single conceivable perception will fit into one class yet close to one: lawfully wedded, normal law wedding, religious wedding, common wedding, living respectively, never wedded, separated, casually differentiated, legitimately divided, widowed, revoked, surrendered and so on.

In ostensible estimation, all perceptions in one class are similar on some property, and vary from the individuals in the other classification on that feature (gender,

military status). We can’t say one classification is better or more terrible or pretty much than others.

2.1.1.2 Type - Ordinal

Ordinal type of estimation utilizes images to arrange perceptions into classifications that are not just totally unrelated and comprehensive. Also, the classifications have some unequivocal relationship between them. Perceptions could be arranged into classifications; for example, longer and shorter, more noteworthy and lesser, quicker and slower, harder and simpler, et cetera. The classes must be comprehensive and totally unrelated.

Most surveys utilize Likert sort things. Case in point, we may get some information about their employment fulfillment. Asking whether an instructor is exceptionally fulfilled, fulfilled, nonpartisan, disappointed or extremely disappointed is utilizing an ordinal type of estimation.

2.1.1.3 Type - Interval

The interim type of estimation makes group perceptions into fundamentally unrelated and comprehensive classes which have some express relationship between them, and the relationship among the classifications is recognized and careful. That is the initial numerical use of numbers.

In the interim type of estimation, a typical and steady unit of estimation is made among the classifications. Cases in point, degrees of temperature are interim measures. A temperature of 76° is one level hotter than a temperature of 75°; similarly, a temperature of 31° is one level colder than a temperature of 32°.

Numbers may be doled out to perceptions in the light of the fact that the relationship among any two classes is thought to be similar with the relationship among numbers in the numerical framework. Case in this point, we can see easily

that 76-1=75 and 31+1=32. Interims among classes are parallel yet they begin from some subjective purpose of birthplace. There exists no significant zero point.

2.1.1.4 Type - Ratio

The proportion type of estimation is similar with the interim degree as the expansion of an important and non-self-assertive zero point. Samples: height, range, speed and velocity. In training, spending plans and number of understudies are estimated on proportion scales.

Variables estimated at a more elevated amount can simply be changed over to a lower level; however, not the other way around. Perceptions of genuine age (proportion-scale) can be caved into classes of more youthful and more seasoned (ordinal-scale), yet age measured basically as more youthful or more seasoned can't be changed over to measures of real age.

2.1.2 Example about Measurement

The attribute "study exertion" may be characterized adroitly as the measure of exertion needed to ace a collection of supplies, involving perusing, turning upward definitions, note-taking, drill, testing toward oneself, et cetera. Each of these segments must be operationalized with a specific end goal to be estimated.

The attribute "study exertion" could be said to comprise of perusing, gazing upward definitions, note-taking, drill and testing toward oneself. Perusing (operational definition): how many hours do you read lecture notes every day? Turning upward definitions (operational definition): by and large, how many words of each course book page do you turn upward? Taking a note (operational definition): how many pages of scripts do you take each course book page? Do you solidify your scripts by taking scripts on your unique notes? Drill (operational definition): how many hours do you study new terms and recipes every day?

2.2 What is Evaluation?

That is the deliberate (methodical) gathering of data about the attributes, exercises and results of administrations or projects to evaluate the degree to which targets have been attained to, recognize required changes and/or settle on choices about future programming.

Evaluation is different from assessment. That is the procedure of social affair and dissecting data about the current status and unmet administration needs of a characterized populace or geographic area. In the connection of the CARE Act, it needs appraisal that includes gathering data about the needs of individuals living with HIV/AIDS (those getting consideration and those not in consideration), recognizing CARE Act and different assets that are accessible to address those issues and deciding the administration holes that need to be tended to.

Evaluation is different from monitoring. That is the continuous methodology of auditing and reporting the project execution, regulatory exercises and consumptions of administration suppliers. The checking methodology incorporates standard audits of every administration supplier's advancement in satisfying contract terms and conditions, the distinguishing proof of territories obliging remedial activity, and catch up to guarantee that restorative activities are effectively actualized.

Evaluation is different from research. That is the procedure of social occasion data to find new learning, test hypotheses or make "truth". The basic role of exploration is to extend the group of learning in a field or teach instead of to advise choice making or enhance administration adequacy.

Evaluation is different from improvement. That is a continuous methodology that includes authoritative individuals in checking and assessing inputs (e.g., staff time, supplies expended), forms (the courses in which administrations are conveyed), yields (number of administration units conveyed) and results so as to persistently enhance administration conveyance (What is evaluation?, n.d.).

2.2.1 Evaluation Types

Assessments can be arranged by center (i.e., what is being assessed) or by their proposed employments. Contingent on the inquiries that you need to reply, an assessment study may concentrate on system structure, administration conveyance courses of action, project yields or system results. When the study's center is dead set, choices must be made on how the consequences of the assessment will be utilized. Will mull over discoveries be utilized to enhance or improve the project (developmental assessment) or to judge the system's general viability (summative assessment)?

2.2.1.1 Structural Evaluation - by Focus

That is an assessment of hierarchical highlights and staff qualities that may impact execution. These are illustrations of some hierarchical highlights that can be assessed incorporate authoritative capacity (i.e., the degree to which administrations are topographically and physically available, socially suitable and accessible at helpful times), installment plans (expense - for - administration versus overseen care), vicinity of a quality change system and accreditation status. Staff qualities may incorporate the number and strength blend of the staff, staff quality (as measured by board certificate or specific preparing) and/or staff-customer proportions.

2.2.1.2 Process Evaluation - by Focus

That is an assessment of the courses in which administrations are conveyed and exercises are performed. Process assessments ask - What is going on and why? How do the parts of the project fit together? How do members see and experience the administrations? - Samples of procedure qualities that can be assessed incorporate:

That is the degree to which administrations are custom-made to the wellbeing - related convictions, mentality, practices and correspondence examples of individual customers.

That is the ability of a wellbeing or social support of producing a coveted result if actualized precisely as outlined and under perfect conditions.

That is the degree to which a wellbeing or social administration meets created proficient benchmarks and client desires.

2.2.1.3 Output Evaluation - by Focus

That is an assessment of the volume of administration conveyed (e.g., number of restorative examinations performed, case administration experiences or lodging arrangements). Cases of yield attributes that can be assessed incorporate:

That is the full cost of delivering or conveying one unit of administration (e.g., one hour of guiding). The normal unit expense is the full cost of giving an administration partitioned by the aggregate number of administration units conveyed.

That is the capacity to deliver a wanted result utilizing insignificant assets. Despite the fact that effectiveness commonly is assessed regarding expense every unit of yield, it can be estimated in different ways. Case in point, a transportation benefit that conveys patients to therapeutic arrangements in the most limited time is the most effective.

That is the relationship between the quantities of administration units given (yields) and the measure of assets devoured (inputs). For instance, a guiding program’s efficiency could be assessed by separating the aggregate number of customer experiences amid a predetermined time period by the aggregate number of staff hours.

2.2.1.4 Outcomes Evaluation - by Focus

That is an assessment of the consequences of an administration or project. Results may be measured for individual customers, a populace or a human services conveyance framework. Cases of result attributes that can be assessed incorporate:

That is the extent to which the objectives and destinations of an administration or project are effectively met.

That is the estimation of an administration or system in respect to its cost. As opposed to money saving advantage examination, which looks at the fiscal expense of a program with the financial advantage, cost-viability investigation does not oblige that expenses and advantages be lessened to an equalizer. For instance, the expense viability of a HIV essential care center could be communicated as taking after - every $1,000 in CARE Act subsidizing for the facility diminished wrong crisis room visits by 0.5 percent.

2.2.1.5 Formative Evaluation - by Intended Uses

That is the accumulation of information for a particular time of time - for the most part amid the start-up or pilot period of a system to enhance usage, take care of unanticipated issues, and guarantee that members are moving toward fancied results. Developmental assessments concentrate on methods for enhancing or upgrading projects instead of judging general viability.

2.2.1.6 Summative Evaluation - by Intended Uses

Gathering of information after a system is well in progress to evaluate its general viability. Summative assessments frequently are directed towards the end of an undertaking period when choices must be made about the continuation or end of a system. The estimation of a particular system can be surveyed, or various projects

can be positioned on an arrangement of criteria; for example, quality, adequacy and expense.

2.3 Software Measurement

Estimation and investigation includes gathering quantitative information about items, methods and tasks and investigating that information to impact your activities and arrangements. Estimation and examination exercises permit you to:

- To portray or addition comprehension of your methodologies, items, assets and situations,
- To assess or to focus your status concerning your arrangements,
- To anticipate by comprehension connections among methods and items so the qualities you watch for a few credits can be utilized to foresee others,
- To enhance by distinguishing barricades, main drivers, inefficiencies and different open doors for development.

In the event that you need responses to these sorts of inquiries, you require estimation and investigation:

- How well would we say we are meeting calendars and spending plans?
- In what capacity would we be able to help our execution change endeavors pay off?
- How does our association's execution contrast with other associations' exhibitions?
- Which programming practices or advancements ought to our association put resources into and what yields would we be able to anticipate?
- Has our execution truly moved forward?

Creating and maintaining an estimation and examination system can be isolated into three essential exercises:

- Anticipating estimation,
- Actualizing your arrangement,
- Enhancing and assessing your endeavors (What is software measurement and analysis?, n.d.).

2.3.1 Planning Measurement

There are likely such a large number of things in your association to estimate that you could undoubtedly get to be overpowered by the opportunities. However, for estimation to be viable, it must be outlined, and focused to backing your business objectives. To start arranging your estimation exercises, ask yourself "What would I like to know or learn?"

2.3.2 Implementing Plan

Estimation regularly is done well and includes esteem. Yet, time after time, estimation:

- is ineffectively incorporated into training and practice,
- stays trying for some associations.

Auditing the issues and accomplishments of different associations can help you execute your arrangement effectively. Numerous associations have imparted their encounters in executing estimation programs.

2.3.3 Evaluating and Improving Efforts

Associations run on information. They utilize it to oversee ventures and the endeavor, decide, and guide change. In the journey to beat the opposition by improving items and forms, quicker and less expensive, having and utilizing brilliant information, and data is info to each key choice. How solid is the information you gather and utilize:

- How great is our information?
- Is it safe to say that we are doing the right things regarding estimation and examination?
- Is it safe to say that we are doing them well?
- How great is the data we create?
- In what capacity would we be able to enhance our estimation forms, and enhance data quality?

2.3.4 Case Study: e-Government and e-Commerce Systems Development

Programming item measurements can be described as outside and inside item measurements. The inside ones are those used to quantify characteristics of the item that can be expressed specifically by exploring the item all alone irrespectively of its conduct. Outer measurements of the item are those used to estimate attributes of the item that can be expressed just as for how the item identifies with its surroundings.

Interior measurements can be separated into three classifications in the light of the item qualities they measure. These classes are: size, many-sided quality and information measurements. To the extent inside item measurements all in all are concerned, it is critical to specify that one of their significant advantages is that they are anything but difficult to systematize, and in this way, information accumulation can be made in a simple, programmed and productive way. Moreover, the estimation conclusions can likewise be analyzed in a customized manner utilizing factual systems, and subsequently results can be made quickly. Instruments; for example, QSUP, GQM robotization and so forth have made inward estimations simple to direct.

Then again, it ought to additionally be noticed that among the inconveniences of inner item estimations, it is the way that they are frequently difficult to clarify. In different cases, the interior amounts measured are not unmistakably connected with the outer quality attributes that ought to be evaluated. Such issues must be settled in

the system of an unmistakable estimation strategy that unites inward and outer measurements, as it will be examined hereinafter.

Taking into account the ISO 9126 standard and other related works, the outside viewpoints influencing programming quality are characterized as functionality, usability, efficiency and reliability. Outside measurements are utilized to quantify these four variables or the characteristics of which these components are framed. This is essential for expressing the distinction between nonexclusive measurements and measurements characterized particularly for e-Commerce frameworks.

Dissimilar to inner measurements (measuring programming inward qualities and coordinating at associating estimations of such attributes with these variables), outer measurements measure these components or their attributes. Such measurements can be made on subjective assessments. Among the methods utilized by outside measurements, they are overviews on client sentiment giving profitable estimations to a fitting programming convenience estimation stock, programming framework usefulness or ease of use. Measures like deformity reports or interim between disappointments are utilized to characterize item unwavering quality while measures like memory utilization are utilized to characterize adequacy.

As already specified, the presentation of outer measurements shows that a clear level of subjectivity is included. Indeed, even measurements that have all the earmarks of being target are regularly considered by some level of subjectivity. Case in point, deformity reports appear to be a strong metric that can be utilized to accurately estimate dependability. Anyhow, it is the time and the level of item use, the client experience and even the client’s inspiration to oversee and present an imperfection report which can impact the quantity of deformity reports put together by a client. Subsequently, such measurements must be explored eagerly and under a system that will contemplate such issues. One of the principle points of interest of outer measurements is that they measure out and out the picked outside item quality highlights by implying that no further investigation or translation is required.

Moreover, outside measurements extraordinarily elevate what is thought to be one of the primary points of e-Government and e-Commerce quality: client fulfillment. Then again, weaknesses and challenges ought to be deliberately considered when choosing to utilize outer measurements. The most essential of which is that such measurements are not objective and subsequently additional exertion is obliged to ensure their objectivity. Furthermore, they are not as financially savvy as inner estimations, and as a rule, it is difficult to oversee estimations because of high blunder rates, particularly in cases that mistake discovery methods have not been utilized amid estimations.

As it was specified anytime recently, interior and outside estimations must be overseen under a very much characterized structure with exact points. Before picking the fitting measurements for any undertaking, a quality manual ought to be performed - this ought to incorporate picking and recording all measurements accessible for utilization in the product creating element. This manual is a principle part of the measurements execution preparation, and incorporates the measurements, the estimation systems, and also, the rules for the usage of measurements, the information investigation and the change activities required for improving the creating methodology of e-Government and e-Commerce frameworks. It ought to additionally be noticed that the quality manual contains all measurements that are accessible independent of how frequently they have been utilized or the accessibility of estimations information from past programming advancement ventures.

For every e-Government or e-Commerce venture, an arrangement of measurements sufficient for this particular undertaking is browsed the quality manual. The standards on which the collection of measurements is based are the uncommon quality components that the undertaking spots accentuation on. This arrangement of measurements is archived - utilizing the rules accessible as a part of the quality manual - and contains the quality arrangement of the particular task. Thusly, an e-Government or e-Commerce venture quality arrangement ought to contain all the measurements, estimation rules and suitable objectives for the task. It is plainly obvious that the venture arrangement of a particular undertaking may

altogether vary from another venture's arrangement, and may utilize a totally diverse arrangement of measurement.

The quality arrangement of every e-Government or e-Commerce task ought to contain inner measurements. This will secure a simple and sensible approach to recognize conceivable reasons for low item quality as this may be seen by the end-clients who may make early restorative move. It ought to likewise contain outer measurements - connected amid alpha or beta testing and post shipment. This will quantify outside quality variables and in addition, the soundness of the inward measurements and estimations results or even adjustment of inner measurements.

The fruitful accumulation of measurements and estimation methods to be incorporated in a substance's quality manual is enormously identified with on the element's development. The selection of advanced strategies and complex measurements by an organization may turn out to be unsuccessful and ineffective in the event that it is not bolstered by years of involvement with measurements and estimations and colossal volumes of information from past undertaking estimations. Programming to create organizations ought to dependably remember this, and ought not to set estimation objectives too high at the early phases of measurements application (Kharytonov, 2012).

2.4 Software Evaluation

The assessment phase of the product advancement procedure requires the customer and engineer to survey the product. They assess against the accompanying inquiries:

- Does this product meet the client necessities?
- Is this fit for reason?

To answer these inquiries, the first points of the product must be assessed against the accompanying criteria:

Robustness: In software engineering, power is the capacity of a PC framework to adapt to slips amid execution. Power can likewise be characterized as the capacity of a calculation to keep working in spite of variations from the norm in info, computations and so forth.

Reliability: Dependability can be characterized as the likelihood that a framework will deliver right yields up to some given time. Reliability is upgraded by highlights that assistance to keep away from, recognize, and repair equipment shortcomings. A solid framework does not quietly proceed with, and convey results that incorporate uncorrected undermined information. Rather, it recognizes, and if conceivable, remedies the defilement, for instance: by retrying an operation for transient (delicate) or discontinuous slips, or else, for uncorrectable blunders, detaching the deficiency and reporting it to larger amount recuperation components (which might failover to repetitive substitution equipment and so forth.) or else by ending the influenced system or the whole framework and reporting the debasement.

Portability: Versatility in abnormal state PC writing computer programs is the convenience of the same programming in distinctive situations. The pre-requirement for versatility is the summed up reflection between the application rationale and framework interfaces. At the point when programming with the same usefulness is created for a few registering stages, convey-ability is the key issue for advancement cost lessening.

Efficiency: In software engineering, algorithmic proficiency is the properties of a calculation which identify with the measure of assets utilized by the calculation. A calculation must be dissected to focus its asset use. Algorithmic proficiency can be considered as comparable to building efficiency for a rehashing or nonstop process. For most extreme proficiency, we wish to minimize asset use. Be that as it may, the different assets (e.g. time, space) cannot be analyzed straightforwardly, so which of two calculations is thought to be more effective regularly relies on upon which measure of productivity is viewed as the most vital, e.g. is the prerequisite for rapid or for least memory use or for some other measure?

Maintainability: Programming support in software building is the alteration of a product item after conveyance to right blames, to enhance execution or other attributes. A typical view of upkeep is that it simply includes settling deformities. Be that as it may, one study demonstrated that more than 80% of support exertion is utilized for non-restorative actions. This recognition is sustained by clients submitting issue reports that truly are usefulness upgrades to the system. More late studies put the bug-settling extent more like 21%.

The product ought to be assessed by customer and designer at all stages all the while, not exactly when the product is finished. For instance;

- How well the customer's issue has been comprehended and ought to be assessed at the examination stage?
- The HCI ought to be assessed at the configuration stage (The software development process - evaluation, n.d.).

CHAPTER THREE
SOFTWARE DEVELOPERS' PERFORMANCE

While it is anything to gauge the product amount, e.g. the measure of lines composed by a software developer, measuring the product quality is more unpredictable. Assessing software programmers on the measure of lines of code they compose won't build up which software developer is more gifted on the other hand notwithstanding meeting expectations harder. For example, composition an entangled calculation may oblige a ton of time, and result in just a couple lines code, making the amount approach out of line.

Assessing software developers by the measure of code they compose would likewise prompt terrible circumstances. A software programmer could be refactoring a ton of the code, making it more agreeable with their venture objectives, which could imply that s/he is just moving or erasing parts of the system, yet s/he wouldn't deliver a ton of new lines of source code. It would be unjustifiable to assess her/him on the measure of source code lines that s/he composes, since s/he would get an awful assessment while s/he would in any case be making a decent showing. Also, when software programmers discover that they are assessed by the measure of code they keep in touch with, it abruptly turns out to be additionally enticing for them to not compose as brief as could be allowed. Circles are abruptly an incredible approach to get a decent assessment, basically by not keeping in touch with them concerning or while pieces yet by duplicate gluing the circling code the same number of times as required.

Still, software developers need to be assessed appropriately. In an instructive circumstance, it is frequently the case that the system gets reviewed rather than the individual software programmers, prompting a uniform evaluation for all the software developers. The great software programmers will get the same review as terrible software developers, which scarcely invigorate them to try their hardest.

Evaluating itself is additionally a method for helping a software developer to advance her/his abilities. By distinguishing the measure of oversights a software programmer sets aside a few minutes a pattern of expanding lapses could be distinguished. This could trigger the software developer's group pioneer to check whether s/he can help the software programmer enhance her/his abilities.

To set up the nature of the code, the program's venture objectives are utilized as the criteria. At the point when an adjustment in the code prompts a superior evaluation of a task objective, the code makes a positive commitment and the other way around. It is however deficient to just take a gander at the program's last state or form, and make an assessment singularly on that. In addition to the fact that it would be difficult to figure out who composed what, it would likewise miss the conceivable changes or decay in the software programmer's capacities.

It is vastly improved to take a gander at more conditions of the system, surveying its agreeability to the venture objectives for every one of those states and connecting them to the software programmer that is capable. The measure of information that this creates can be tremendous, contingent upon the quantity of variants and the measure of criteria.

CHAPTER FOUR
LITERATURE REVIEW

In the result of our literature search about measurement and evaluation of software developers' performance, we have seen that seventeen articles with scientific index and one master thesis which have been completed in 2008 have been related to our work:

"Baggelaar, H. (2008). Evaluating programmer performance visualizing the impact of programmers on project goals. M.Sc Thesis, University of Amsterdam, Amsterdam." Based on this thesis, twenty two meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Balijepally, V., Nerur, S., & Mahapatra, R. (2012). Effect of task mental models on software developer's performance: An experimental investigation. 45th Hawaii International Conference on System Science, IEEE, 5442 - 5451." Based on this article, eight meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Calikli, G., & Bener, A. (2013). An algorithmic approach to missing data problem in modeling human aspects in software development. Proceedings of the 9th International Conference on Predictive Models in Software Engineering, ACM, Article No. 10." Based on this article, four meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Calikli, G., & Bener, A. (2010). Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance. Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, Article No. 10." Based on this article, four meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

“Chilton, M. A., Hardgrave, B. C., & Armstrong, D. J. (2010). Performance and strain levels of it workers engaged in rapidly changing environments: A person-job fit perspective. ACM SIGMIS, 8 - 35.” Based on this article, twenty two meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Duarte, C. B., Faria, J. P., & Raza, M. (2012). PSP PAIR: Automated personal software process performance analysis and improvement recommendation. Eighth International Conference on the Quality of Information and Communcations Technology, IEEE, 131 - 136.” Based on this article, ten meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Ehrlich, K., & Cataldo, M. (2012). All-for-one and one-for-all?: A multi-level analysis of communication patterns and individual performance in geographically distributed software development. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, 945 - 954.” Based on this article, ten meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Gallivan, M. J. (1998). The influence of system developers’ creative style on their attitudes toward and assimilation of a software process innovation. Proceedings of the Thirty-First Hawaii International Conference on System Sciences, IEEE, 435 - 444.” Based on this article, eighteen meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Hall, T., Wilson, D., Rainer, A., & Jagielska, D. (2007). The neglected technical skill? Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce, 196 - 202.” Based on this article, twenty six meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Kelly, B., & Haddad, H. M. (2012). Metric techniques for maintenance programmers in a maintenance ticket environment. Journal of Computing Sciences in Colleges, ACM, 28 (2), 170 - 178.” Based on this article, three meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Lee, K., Joshi, K., & Kim, Y. (2008). Person-job fit as a moderator of the relationship between emotional intelligence and job performance. Proceedings of the 2008 ACM SIGMIS CPR Conference on Computer Personnel Doctoral Consortium and Research, 70 - 75.” Based on this article, twelve meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Ramler, R., Klammer, C., & Natschläger, T. (2010). The usual suspects: A case study on delivered defects per developer. Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Article No. 48.” Based on this article, three meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Sawyer, S., & Guinan, P. J. (1998). Software development: Processes and performance. IBM Systems Journal, 37 (4), 552 - 569.” Based on this article, forty four meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Schröter, A., Aranda, J., Damian, D., & Kwan, I. (2012). To talk or not to talk: Factors that influence communication around changesets. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, 1317 - 1326.” Based on this article, twenty two meaningful criteria about measurement and evaluation of software developers’ performance have been extracted.

“Thing, C. (2008). The application of the function point analysis in software developers’ performance evaluation. 4th International Conference on Wireless

Communications, Networking and Mobile Computing, IEEE, 1 - 4." Based on this article, fourteen meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Wang, Y., & Zhang, M. (2010). Penalty policies in professional software development practice: A multi-method field study. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 39 - 47." Based on this article, fourteen meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Westermann, D. (2012). A generic methodology to derive domain-specific performance feedback for developers. 34th International Conference on Software Engineering, IEEE, 1527 - 1530." Based on this article, seven meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

"Zhang, S., Wang, Y., & Xiao, J. (2008). Mining individual performance indicators in collaborative development using software repositories. 15th Asia-Pacific Software Engineering Conference, IEEE, 247 - 254." Based on this article, thirteen meaningful criteria about measurement and evaluation of software developers' performance have been extracted.

After studying on those articles, we have created a survey with sixty three criteria about measurement and evaluation of software developers' performance. Our aim is to minimize some bad things in software crisis in software engineering, such as, over-budget, over-time, low quality, not meeting requirements, inefficiency by the contribution of the results of the survey. If our critearia set which have been formed based on the survey results achieve the goal of us, maybe, the term software crisis in software engineering will disappear.

CHAPTER FIVE
CRITERIA ABOUT MEASUREMENT AND EVALUATION OF SOFTWARE DEVELOPERS' PERFORMANCE

The survey has included sixty three criteria and five main sections about measurement and evaluation of software developers' performance: personal information, program development, design and management, program completion process and working place atmosphere. This survey has been prepared on the google document form application on internet, so the participants could take this survey on internet. In total, one hundred and five software developers have conducted the survey, but ten of them have sent the survey with empty answers and fourteen of them have answered the questions in the survey illogically. For instance, someone has declared while s/he has written one thousand comment lines, s/he has written one code line for a day while developing a program, or someone has declared while s/he has developed one code line, s/he has done one thousand build operations for a day while developing a program. After deleting those participants, we have seen that eighty one participants have answered all the questions logically in the survey and they have worked at fifty five different companies. In the following, the questions in the survey and their answers given by the participants have been explained.

5.1 What is Your Age?

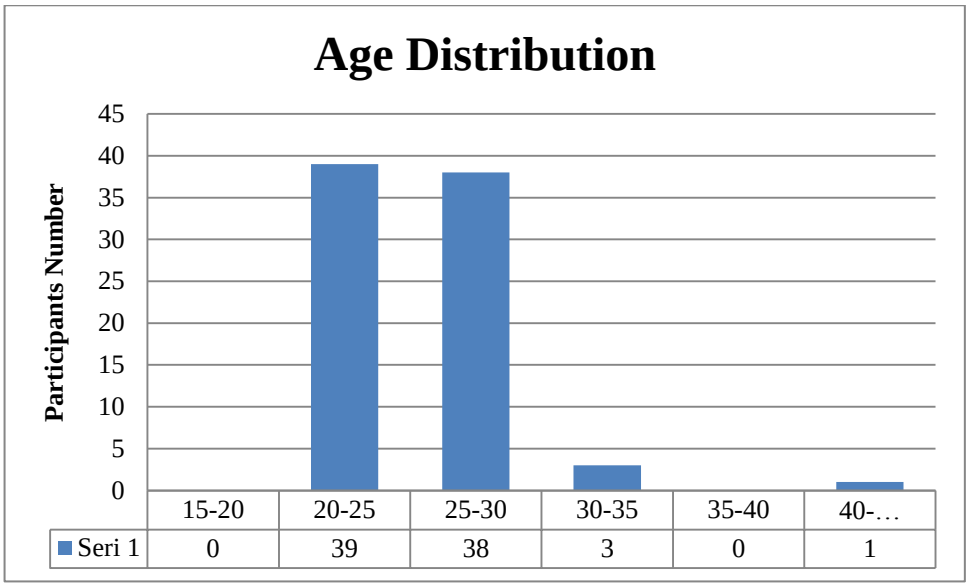

Figure 5.1 Age distribution

Based on the "Figure 5.1", among the participants who have taken the survey, about 48% of them are 20 to 25 years old, about 47% of them are 25 to 30 years old, 3.5% of them are 30 to 35 years old, and only 1 of them is more than 40 years old. However, although "15-20" and "35-40" age options have been given to the participants, who have taken this survey, in this question, none of them has chosen these age groups. In addition, based on the diagram, it can be said that almost all of the participants, who have taken the survey, are between 20 and 30 years old.

5.2 What is Your Education Level? (The Last Graduated Level)

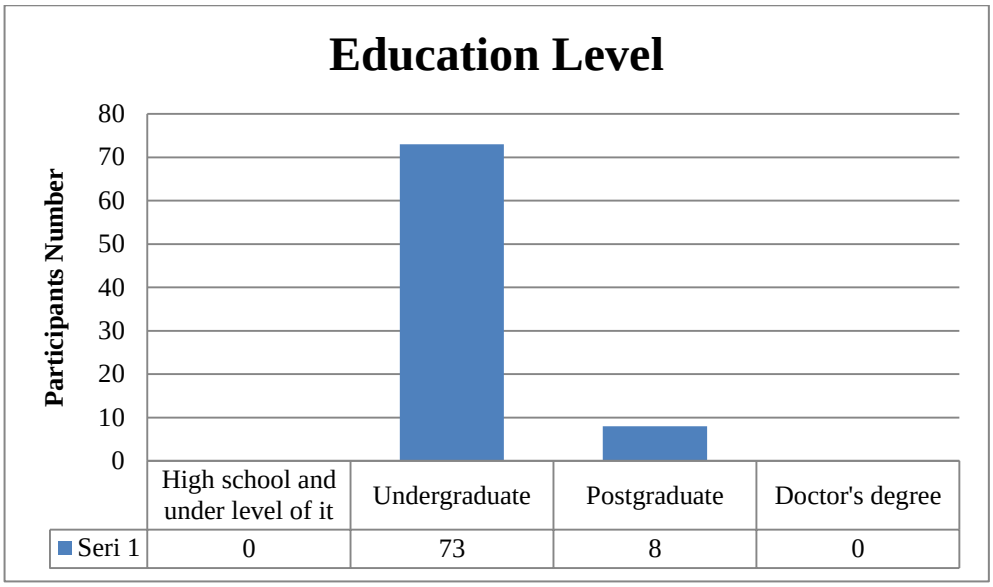

Figure 5.2 Education level

Based on the “Figure 5.2”, among the participants who have taken the survey, all of them have graduated from a university, and also, about 10% of them have taken their master degrees. However, although “High school and under level of it” and “Doctor’s degree” education level options have been given to the participants, who have taken this survey, in this question, none of them has chosen these education level groups. In addition, based on the diagram, it can be said that all of the participants, who have taken the survey, have a bachelor’s degree.

5.3 What is Your Education Department?

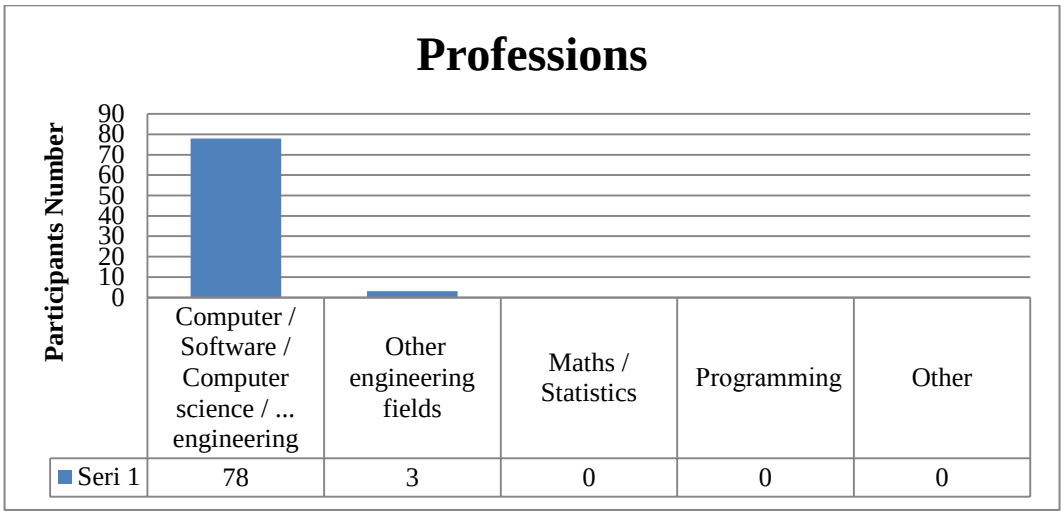

Figure 5.3 Professions

Based on the "Figure 5.3", among the participants who have taken the survey, about 96% of them have studied at the department of computer engineering or software engineering or computer science, and about 4% of them have studied at the department of other engineering fields. However, although "Maths/Statics", "Programming" and "Other" education department options have been given to the participants, who have taken this survey, in this question, none of them has chosen these education department groups. In addition, based on the diagram, it can be said that almost all of the participants, who have taken the survey, have studied at the department of computer engineering or software engineering or computer science at a university.

5.4 How many Code Lines do You Write without Comment Lines for a Day?

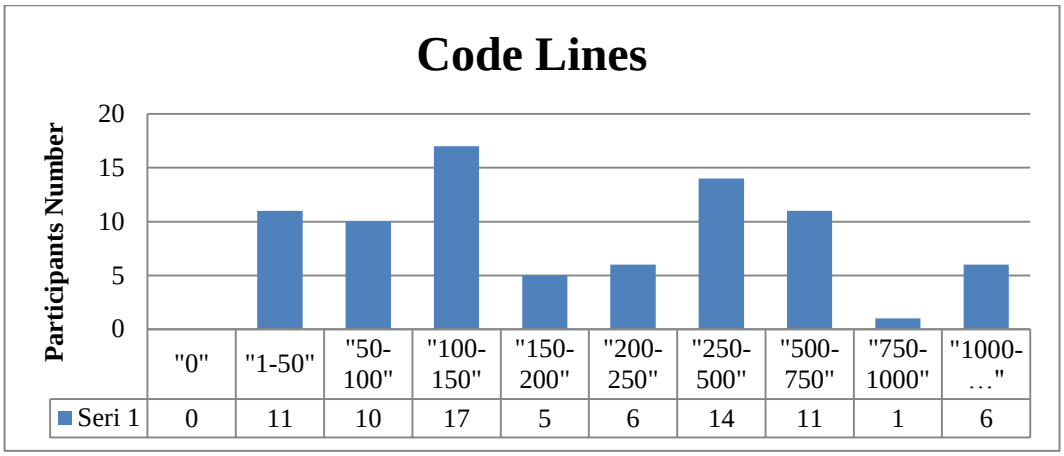

Figure 5.4 Code lines

Based on the “Figure 5.4”, among the participants who have taken the survey, about 13.5% of them write 1 to 50 code lines, about 12.5% of them write 50 to 100 code lines, about 21% of them write 100 to 150 code lines, about 6% of them write 150-200 code lines, about 7.5% of them write 200-250 code lines, about 17% of them write 250-500 code lines, about 13.5% of them write 500-750 code lines, about 1.5% of them write 750-1000 code lines, and about 7.5% of them write more than 1000 code lines without any comment lines for a day on average while developing a program. In addition, the diagram shows that all participants, who have taken this survey, write codes in different code lines groups every day while writing a program. Moreover, the average code lines amount which the participants, who have taken the survey, write for a day is roughly 262 while developing a program.

5.5 Build Operation

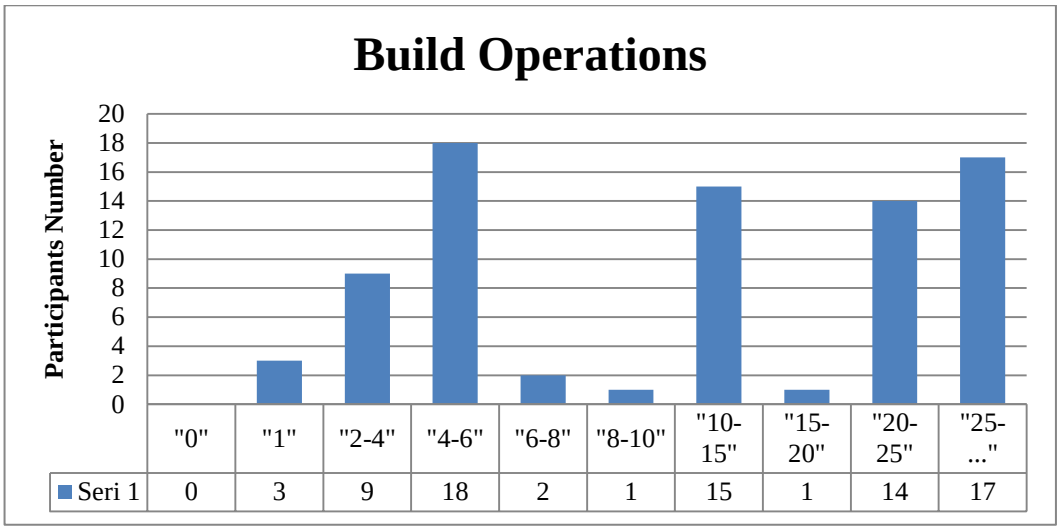

Figure 5.5 Build operations

Based on the “Figure 5.5”, among the participants who have taken the survey, about 4% of them do only 1 build operation, about 11% of them do 2 to 4 build operations, about 22% of them do 4 to 6 build operations, about 2.5% of them do 6 to 8 build operations, about 1% of them do 8 to 10 build operations, about 18.5% of them do 10 to 15 build operations, about 1% of them do 15 to 20 build operations, about 17.5% of them do 20 to 25 build operations, and about 21% of them do more than 25 build operations for a day on average while developing a program. In addition, the diagram shows that all participants do at least 1 build operation every day while writing a program. Also, only 1 of them - the rest of the participants - hasn’t answered this question in this survey based on the given options by writing “once a week”. Moreover, the average build operations number which the participants, who have taken the survey, do for a day is roughly 19 while developing a program.

5.6 How many Times do You Change the Code, which You Have Written, for a Day?

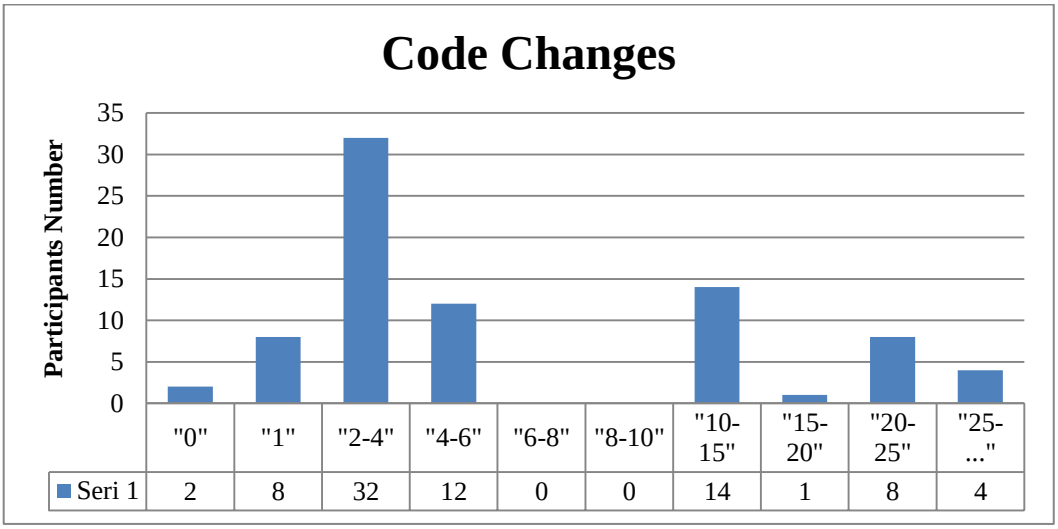

Figure 5.6 Code changes

Based on the “Figure 5.6”, among the participants who have taken the survey, about 2.5% of them don’t do any code changes, about 10% of them do only 1 code change, about 39.5% of them do 2 to 4 code changes, about 15% of them do 4 to 6 code changes, about 17% of them do 10 to 15 code changes, about 1% of them do 15 to 20 code changes, about 10% of them do 20 to 25 code changes, and about 5% of them do more than 25 code changes for a day on average while developing a program. In addition, the diagram shows that none of the participants, who have taken this survey, has chosen the code changes groups 6 to 8 and 8 to 10 in this question. Furthermore, the average code changes number which the participants, who have taken the survey, do for a day is roughly 8 while writing a program.

5.7 How many Comment Lines do You Write for a Day on Average?

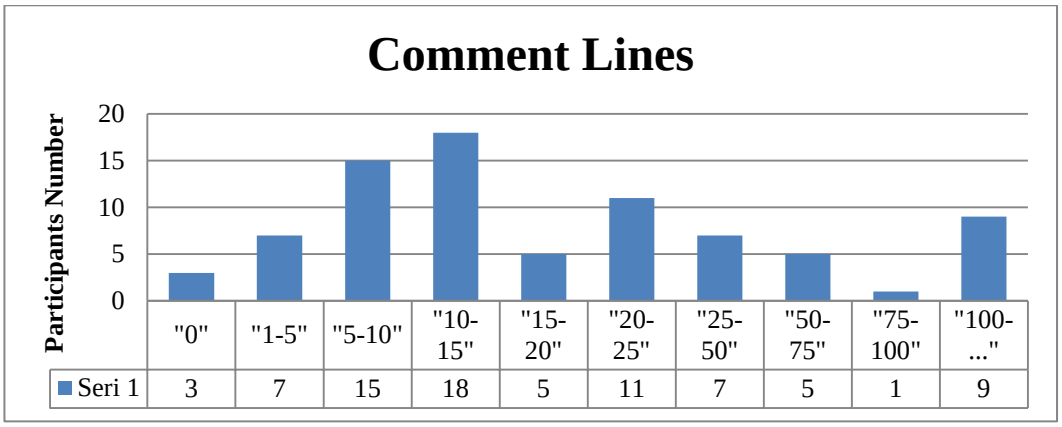

Figure 5.7 Comment lines

Based on the "Figure 5.7", among the participants who have taken the survey, about 4% of them don't write any comments, about 8.5% of them write 1 to 5 comment lines, about 18.5% of them write 5 to 10 comment lines, about 22.5% of them write 10 to 15 comment lines, about 6% of them write 15 to 20 comment lines, about 13.5% of them write 20 to 25 comment lines, about 8.5% of them write 25 to 50 comment lines, about 6% of them write 50 to 75 comment lines, about 1.5% of them write 75 to 100 comment lines, and 11% of them write more than 100 comment lines for a day on average while developing a program. In addition, the average comment lines amount which the participants, who have taken the survey, write for a day is roughly 37 while writing a program.

5.8 How many Classes do You Write while Developing a Program for a Day?

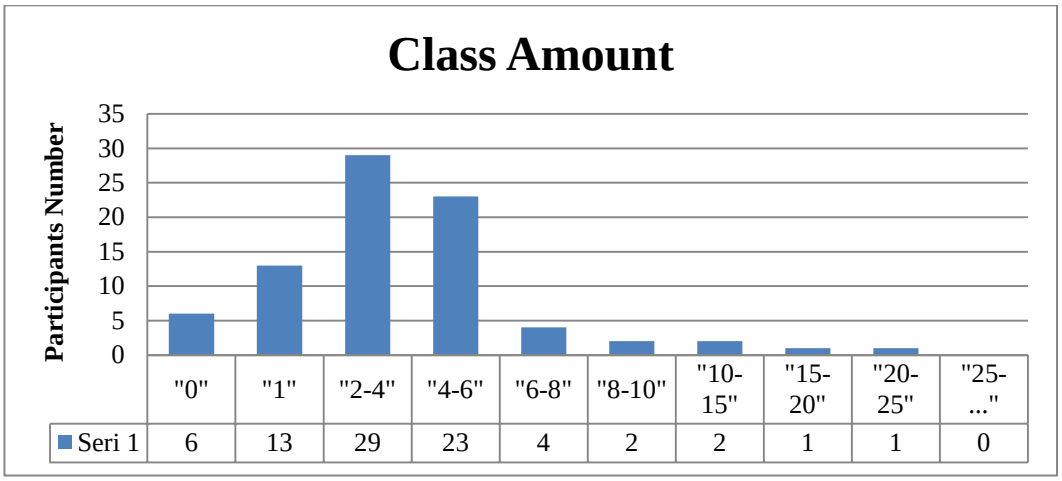

Figure 5.8 Class amount

Based on the “Figure 5.8”, among the participants who have taken the survey, about 7.5% of them don’t develop any classes, about 16% of them develop only 1 class, about 36% of them develop 2 to 4 classes, about 28.5% of them develop 4 to 6 classes, about 5% of them develop 6 to 8 classes, about 2.5% of them develop 8 to 10 classes, about 2.5% of them develop 10 to 15 classes, about 1% of them develop 15 to 20 classes, and about 1% of them develop 20 to 25 classes for a day on average while developing a program. In addition, the diagram shows that none of the participants, who have taken this survey, has chosen the class amount group more than 25 classes in this question. Moreover, the diagram shows that most of the participants, who have taken this survey, develop less than 6 classes for a day on average while writing a program. Also, the average class number which the participants, who have taken the survey, develop in a program for a day is roughly 4.

5.9 How many Methods do You Write in the Program for a Day on Average?

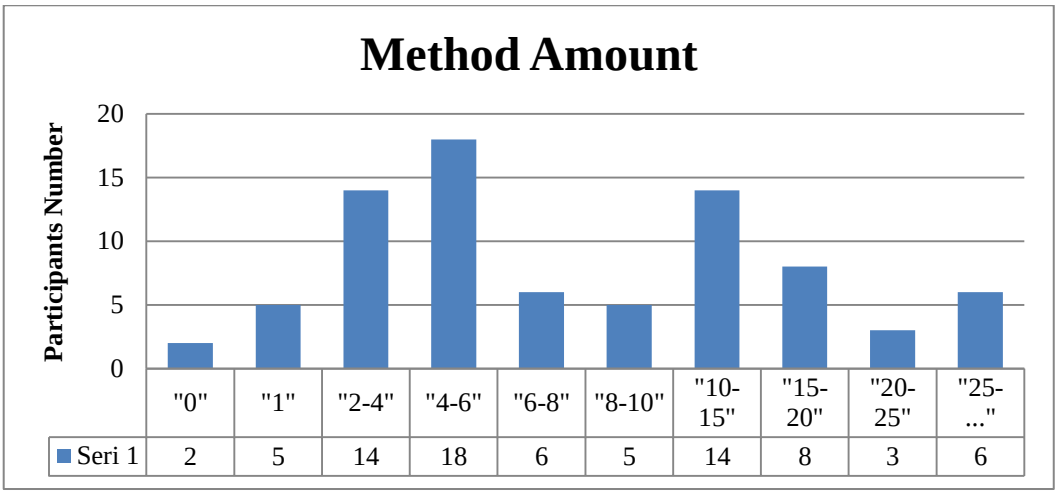

Figure 5.9 Method amount

Based on the “Figure 5.9”, among the participants who have taken the survey, about 2.5% of them don’t develop any methods, about 6% of them develop only 1 method, about 17% of them develop 2 to 4 methods, about 22.5% of them develop 4 to 6 methods, about 7.5% of them develop 6 to 8 methods, about 6% of them develop 8 to 10 methods, about 17% of them develop 10 to 15 methods, about 10% of them develop 15 to 20 methods, about 4% of them develop 20 to 25 methods, and 7.5% of them develop more than 25 methods for a day on average while developing a program. In addition, the average method number which the participants, who have taken the survey, develop in a program for a day is roughly 12.

5.10 How many Modules are You able to Complete while Developing a Program?

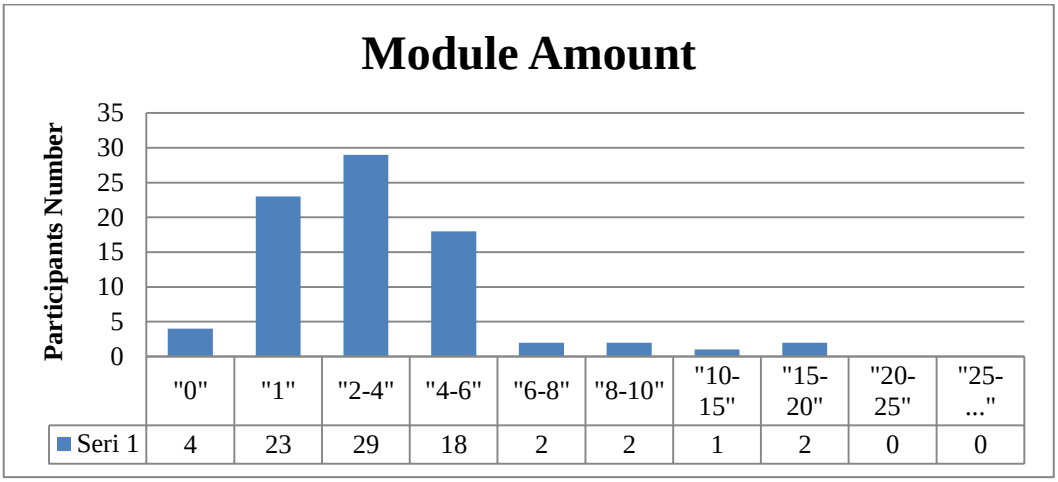

Figure 5.10 Module amount

Based on the “Figure 5.10”, among the participants who have taken the survey, about 5% of them don’t develop any modules, about 28.5% of them develop only 1 module, about 36% of them develop 2 to 4 modules, about 22% of them develop 4 to 6 modules, about 2.5% of them develop 6 to 8 modules, about 2.5% of them develop 8 to 10 modules, about 1% of them develop 10 to 15 modules, and about 2.5% of them develop 15 to 20 modules for a day on average while developing a program. In addition, the diagram shows that none of the participants, who have taken this survey, has chosen the module amount groups 20 to 25 modules and more than 25 modules in this question. Also, the diagram shows that most of the participants, who have taken this survey, develop less than 6 modules for a day on average while writing a program. Moreover, the average module number which the participants, who have taken the survey, develop in a program for a day is roughly 3.

5.11 How many Bugs are You able to Determine in the Program, for a Day?

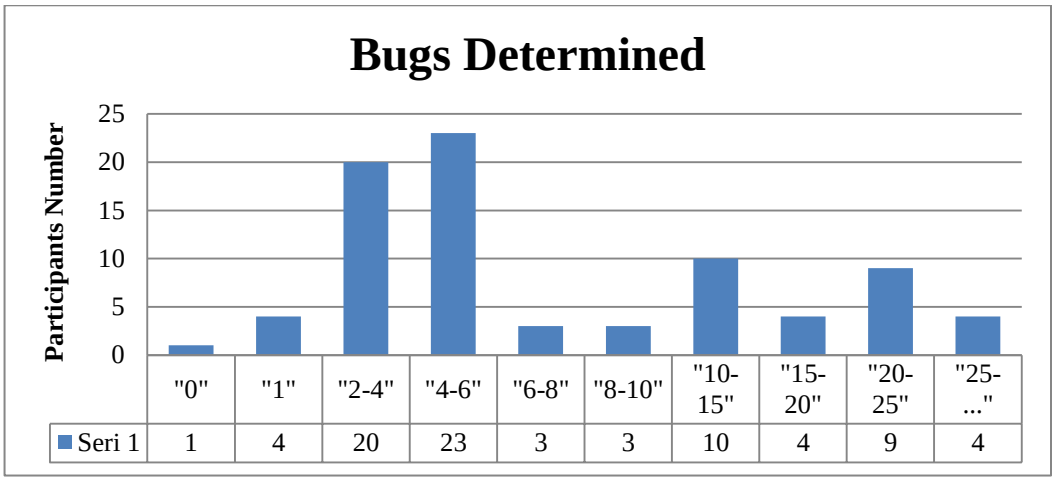

Figure 5.11 Bugs determined

Based on the “Figure 5.11”, among the participants who have taken the survey, about 1% of them don’t determine any bugs, about 5% of them determine only 1 bug, about 24.5% of them determine 2 to 4 bugs, about 28.5% of determine 4 to 6 bugs, about 4% of them determine 6 to 8 bugs, about 4% of them determine 8 to 10 bugs, about 12% of them determine 10 to 15 bugs, about 5% of them determine 15 to 20 bugs, about 11% of them determine 20 to 25 bugs, and 5% of them determine more than 25 bugs for a day on average while developing a program. In addition, the average bugs number which the participants, who have taken the survey, determine in a program for a day is roughly 9.

5.12 How many Bugs are You able to Fix in the Program, for a Day on Average?

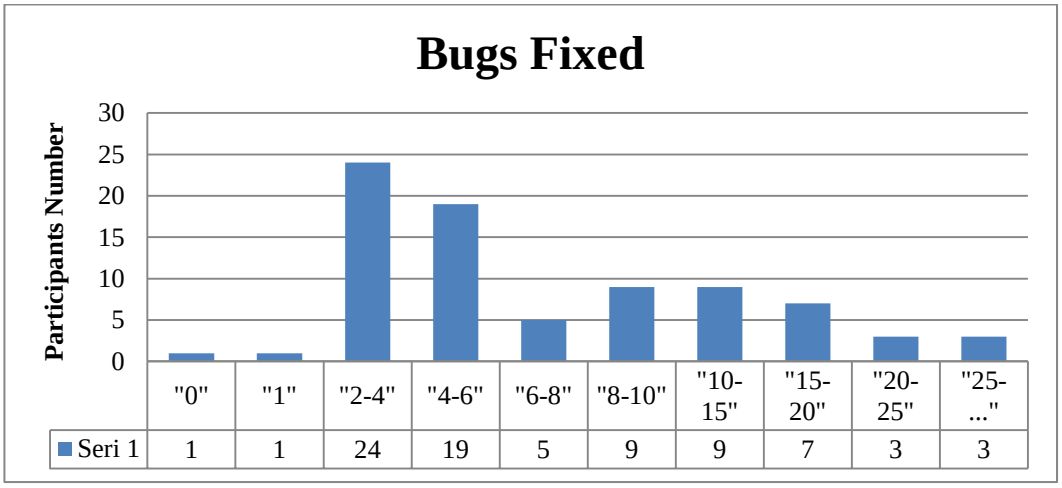

Figure 5.12 Bugs fixed

Based on the “Figure 5.12”, among the participants who have taken the survey, about 1% of them don’t fix any bugs, about 1% of them fix only 1 bug, about 30% of them fix 2 to 4 bugs, about 23.5% of them fix 4 to 6 bugs, about 6% of them fix 6 to 8 bugs, about 11% of them fix 8 to 10 bugs, about 11% of them fix 10 to 15 bugs, about 8.5% of them fix 15 to 20 bugs, about 4% of them fix 20 to 25 bugs, and 4% of them fix more than 25 bugs for a day on average while developing a program. In addition, the average bugs number which the participants, who have taken the survey, fix in a program for a day is roughly 8.

5.13 What is the Average Time of the Error Correction?

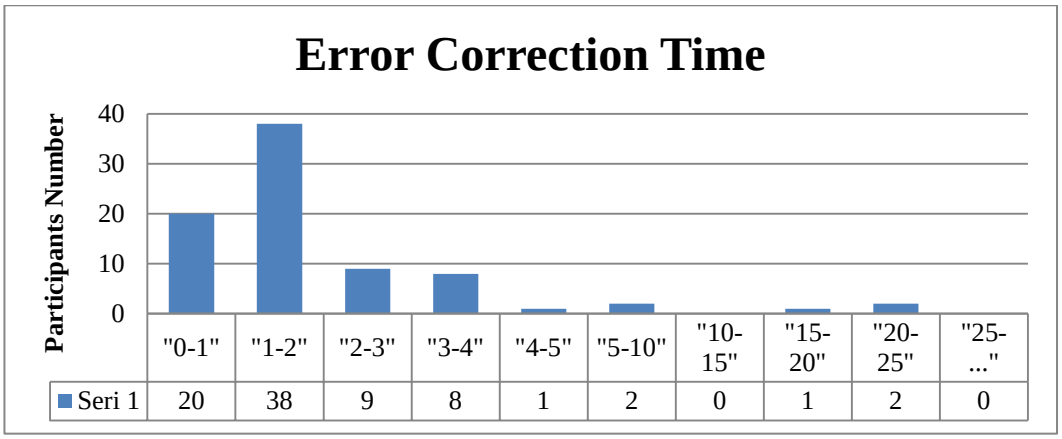

Figure 5.13 Error correction time

Based on the “Figure 5.13”, among the participants who have taken the survey, about 25% of them fix the bugs, which they have determined in the program, in 1 hour, about 47% of them fix the bugs, which they have determined in the program, in 1 to 2 hours, about 11% of them fix the bugs, which they have determined in the program, in 2 to 3 hours, about 10% of them fix the bugs, which they have determined in the program, in 3 to 4 hours, about 1% of them fix the bugs, which they have determined in the program, in 4 to 5 hours, about 2.5% of them fix the bugs, which they have determined in the program, in 5 to 10 hours, about 1% of them fix the bugs, which they have determined in the program, in 15 to 20 hours, and about 2.5% of them fix the bugs, which they have determined in the program, which they have developed, in 20 to 25 hours. In addition, the diagram shows that none of the participants, who have taken this survey, has chosen the error correction time groups 10 to 15 hours and more than 25 hours in this question. Moreover, the diagram shows that most of the participants fix the problem in a program which they have developed less than 4 hours on average. Also, the average error correction time which the participants, who have taken the survey, fix the bugs, which they have determined in a program, which they have developed is roughly 2 hours.

5.14 How many Bugs which You have Fixed Before Give Error Again?

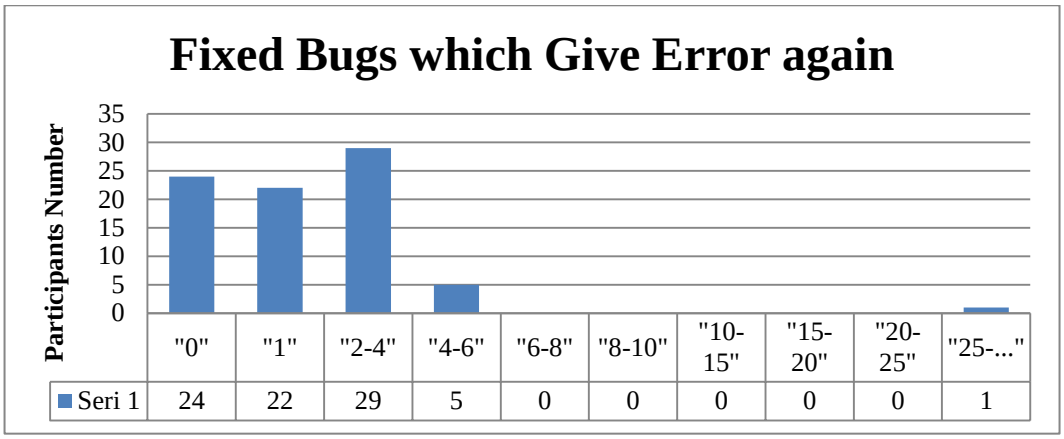

Figure 5.14 Fixed bugs which give error again

Based on the "Figure 5.14", among the participants who have taken the survey, about 30% of them don't face with same bugs, which they have fixed before, again, about 27% of them face with same only 1 bug, which they have fixed before, again, about 36% of them face with same 2 to 4 bugs, which they have fixed before, again, about 6% of them face with same 4 to 6 bugs, which they have fixed before, again, and 1% of them face with same more than 25 bugs, which they have fixed before, again while developing a program for a day on average. In addition, the diagram shows that none of the participants, who have taken this survey, has chosen the bugs groups 6 to 8 bugs, 8 to 10 bugs, 10 to 15 bugs, 15 to 20 bugs and 20 to 25 bugs in this question. Also, the diagram shows that almost all of the participants face with same less than 4 bugs on average in the program. Moreover, the average bugs number which the participants, who have taken the survey, have fixed before in a program, which they have developed, give error again is roughly 2.

5.15 How many Unit Tests do You Write for a Day on Average?

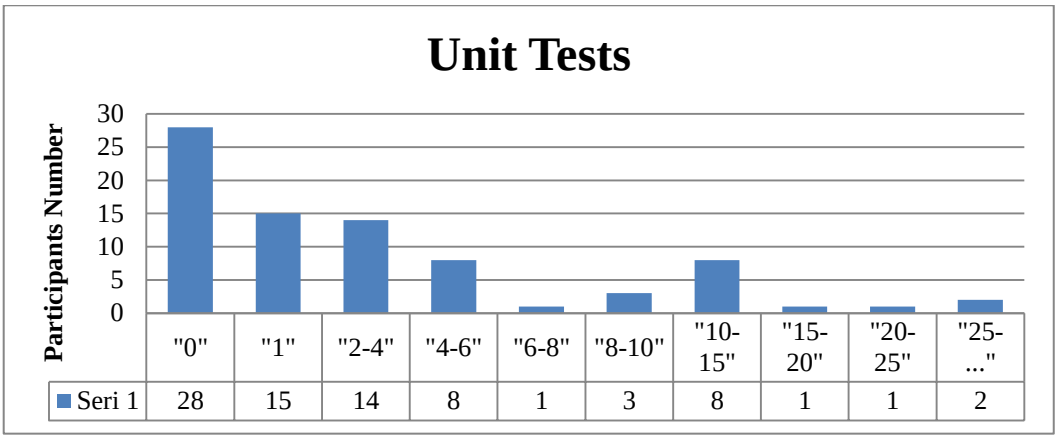

Figure 5.15 Unit tests

Based on the “Figure 5.15”, among the participants who have taken the survey, about 34.5% of them don’t write any unit test cases, about 18.5% of them write only 1 unit test case, about 17.5% of them write 2 to 4 unit test cases, about 10% of them write 4 to 6 unit test cases, about 1% of them write 6 to 8 unit test cases, about 4% of them write 8 to 10 unit test cases, about 10% of them write 10 to 15 unit test cases, about 1% of them write 15 to 20 unit test cases, about 1% of them write 20 to 25 unit test cases, and 2.5% of them write more than 25 unit test cases for a day on average while developing a program. Moreover, the diagram shows that most of the participants, who have taken this survey, write less than 6 unit test cases for a day on average while writing a program. In addition, the average unit test cases number which the participants, who have taken the survey, write for a day is roughly 4 while developing a program.

5.16 Do You Reuse any Code Parts while Writing (Developing) a Program?

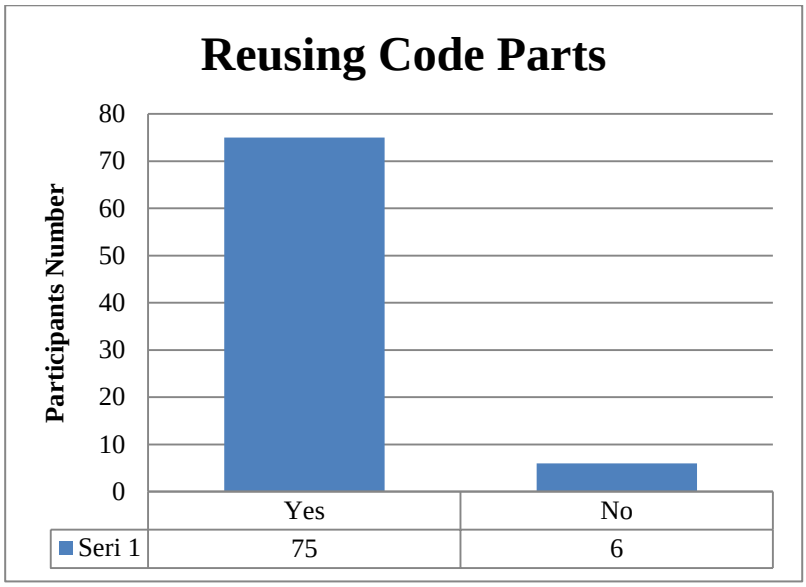

Figure 5.16 Reusing code parts

Based on the "Figure 5.16", among the participants who have taken the survey, about 92.5% of them reuse some code parts which have been developed before by themselves or someone else; however, about 7.5% of them don't do that while developing a program. In addition, the diagram shows that most of the participants, who have taken this survey, do the operation of reusing some code parts which have been developed before by themselves or someone else while writing a program.

5.17 Transmission

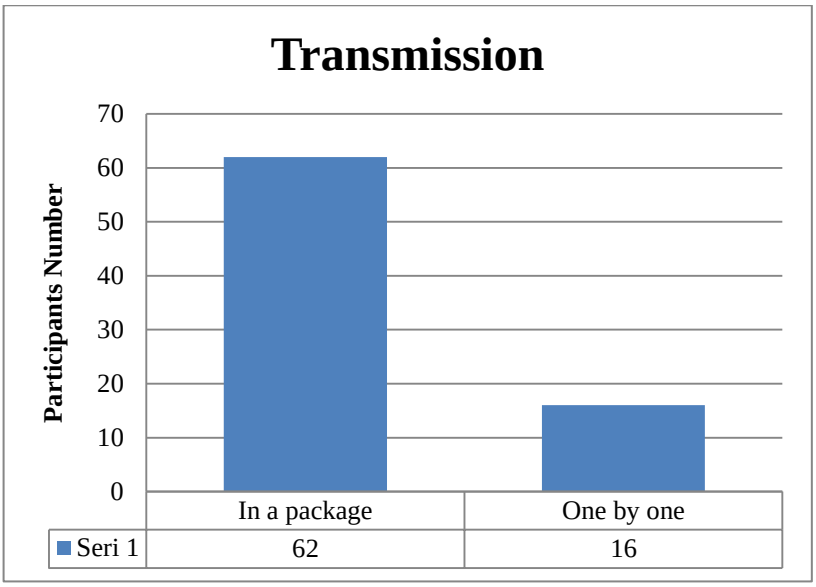

Figure 5.17 Transmission

Based on the “Figure 5.17”, among the participants who have taken the survey, about 76.5% of them transmit the modules, which they have developed, to the user in a package by being combined; however, about 19.5% of them transmit the modules, which they have developed, to the user directly one by one. Also, the rest of the participants - 3 people - transmit the modules, which they have developed, to the user in both way - sometimes “in a package”, but sometimes “one by one” - based on the projects. In addition, the diagram shows that most of the participants, who have taken this survey, deliver the modules, which they have developed, to the user in a package by being combined.

5.18 Are There any Rules which Specify the Names while Developing a Program?

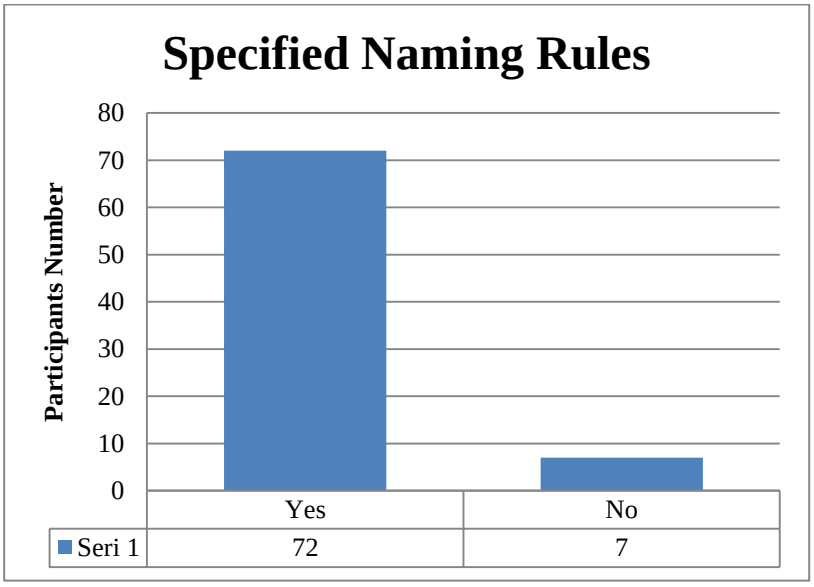

Figure 5.18 Specified naming rules

Based on the "Figure 5.18", among the participants who have taken the survey, about 89% of them use some rules which specify the names of variables, methods and classes; however, about 8.5% of them don't use these rules while developing a program. In addition, the rest of the participants - 2 people - sometimes use, but sometimes don't use those rules based on the projects while writing a program. Moreover, the diagram shows that most of the participants, who have taken this survey, use some rules which specify the names of variables, methods and classes while developing a program.

5.19 Do You Use any Software Development Method while Developing a Program?

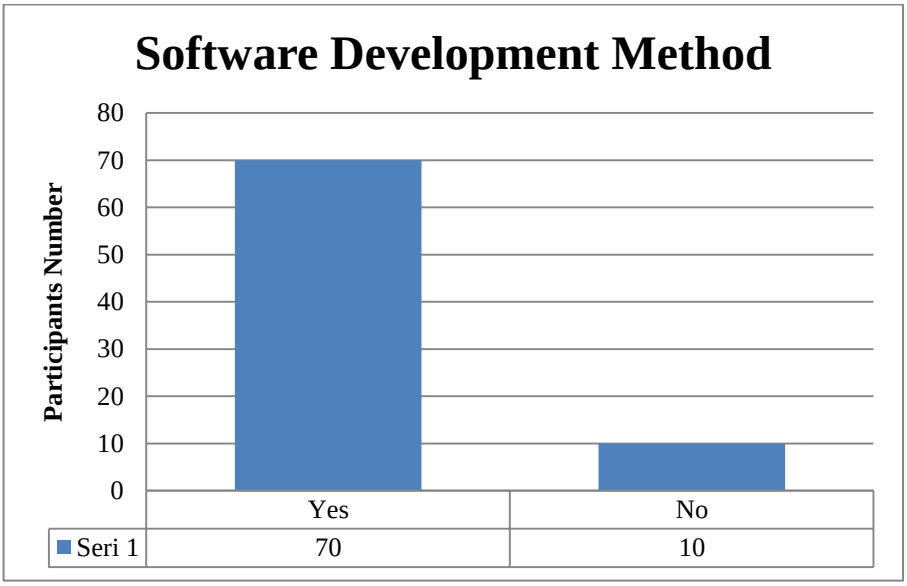

Figure 5.19 Software development method

Based on the "Figure 5.19", among the participants who have taken the survey, about 86.5% of them use some software development method; however, about 12% of them don't use any software development method while developing a program. Moreover, 1 of them - the rest of the participants - sometimes use, but sometimes don't use a software development method while writing a program based on the projects. In addition, the diagram shows that most of the participants, who have taken this survey, use a software development method while developing a program.

5.20 Do You Use any Automated Tool to Ensure Following Your Software Development Method?

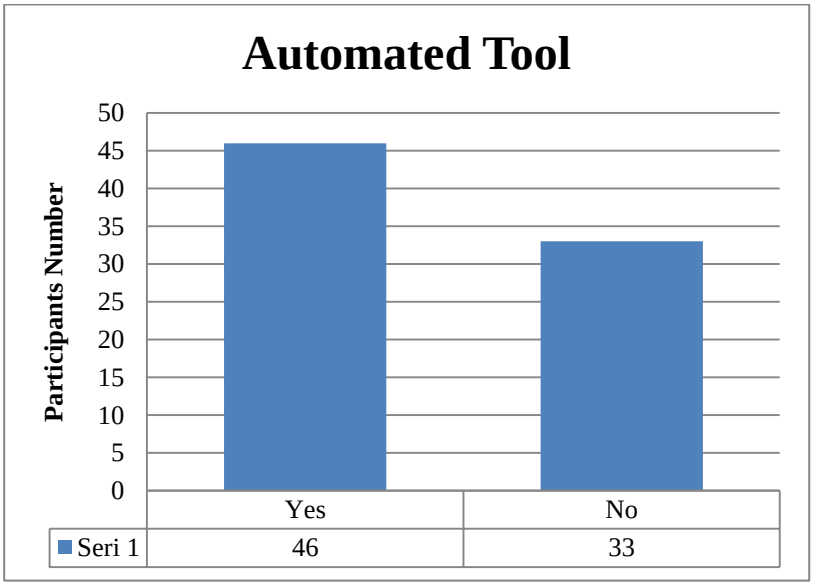

Figure 5.20 Automated tool

Based on the "Figure 5.20", among the participants who have taken the survey, about 57% of them use some automated tool; however, about 40.5% of them don't use any automated tool to ensure their software development methods while developing a program. Furthermore, 2 of them - the rest of the participants - sometimes use, but sometimes don't use an automated tool based on the projects to ensure following their software development methods which are used while writing a program.

5.21 Do You Use any Methods or Models which have been Determined before?

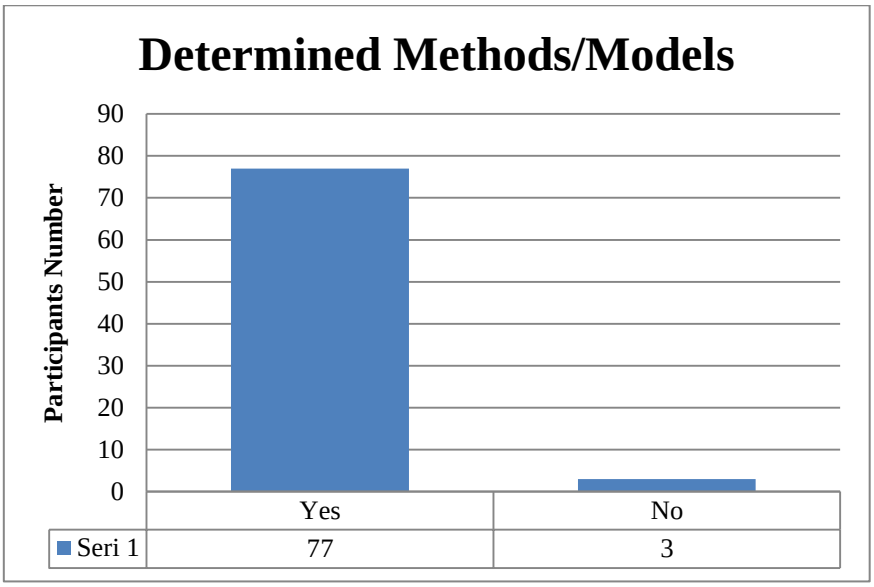

Figure 5.21 Determined methods/models

Based on the "Figure 5.21", among the participants who have taken the survey, about 95% of them use some methods or models which have been determined before while developing a program; however, about 3.5% of them don't use these. Also, 1 of them - the rest of the participants - sometimes uses, but sometimes doesn't use some methods or models which have been determined before while writing a program based on the projects. In addition, the diagram shows that almost all of the participants, who have taken this survey, use some methods or models which have been determined before while developing a program.

5.22 Is There a Shared Code Storage or a Library for You?

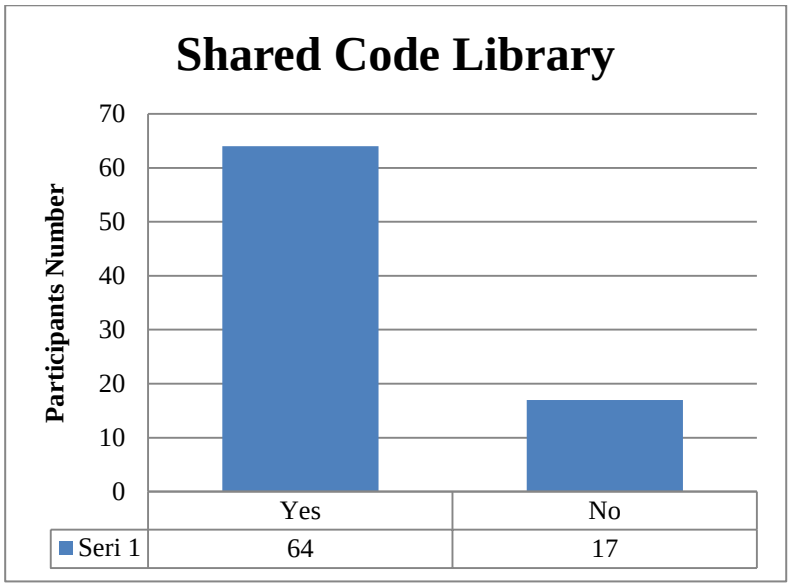

Figure 5.22 Shared code library

Based on the "Figure 5.22", among the participants who have taken the survey, about 79% of them use a code library or storage, but about 21% of them don't use any code library or storage while developing a program. In addition, the diagram shows that most of the participants, who have taken this survey, use a code library or storage while writing a program.

5.23 Code Changes

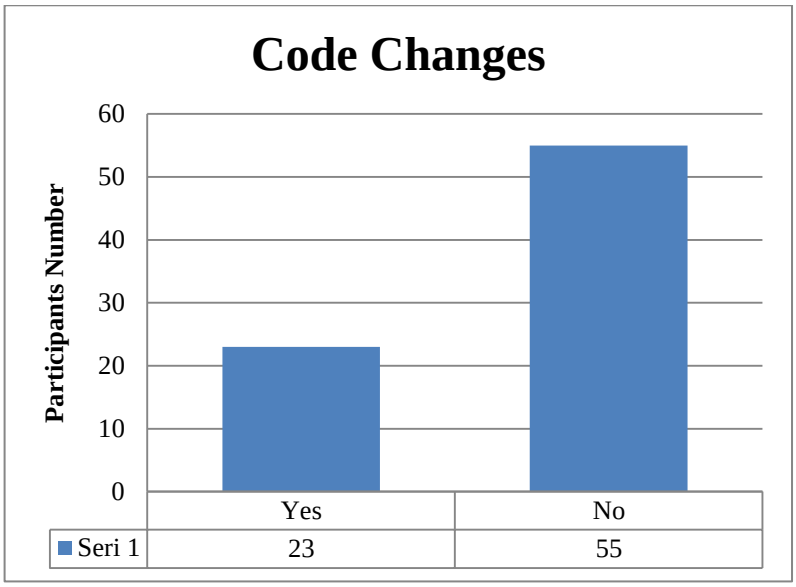

Figure 5.23 Code changes

Based on the "Figure 5.23", among the participants who have taken the survey, about 28% of them change the codes, which they have written before, frequently; however, about 68% of them don't change them frequently while developing a program. Moreover, the needs of 3 of them - the rest of the participants - about change of the codes, which they have written before, change based on the projects while writing a program.

5.24 What is the Degree of Inheritance Depth while Developing a Program?

In item arranged programming, legacy is the point at which an article or a class is in view of another protest or class by utilizing same execution (acquiring from a class) and determining usage to keep up the same conduct. It is a system for code reuse, and to permit autonomous augmentations of the first programming by means of open classes and interfaces. The connections of items or classes through legacy offer an ascent to a chain of commands (Inheritance (object-oriented programming), n.d.).

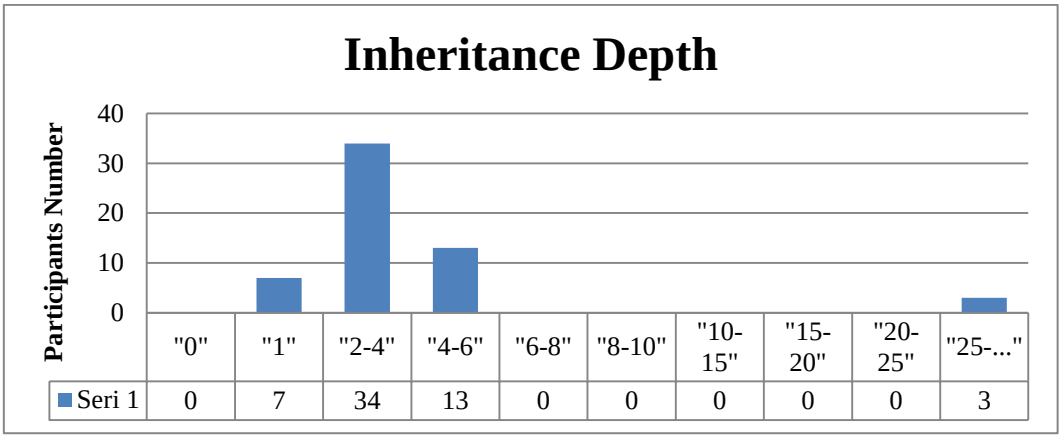

Figure 5.24 Inheritance depth

Based on the “Figure 5.24”, among the participants who have taken the survey, about 8.5% of them develop inheritance relationship with only 1 degree, about 42% of them develop inheritance relationship with 2 to 4 degrees, about 16% of them develop inheritance relationship with 4 to 6 degrees, and about 4% of them develop inheritance relationship with more than 25 degrees while developing a program. In addition, 23 of the participants haven’t answered this question so that we may understand from that they don’t use any inheritance relationship while writing a program, and also, 1 of them - the rest of the participants - has answered this question by writing “changes based on the projects” in the survey. Furthermore, the diagram shows that most of the participants, who have taken this survey, develop inheritance relationship with less than 6 degrees while developing a program. Moreover, the average inheritance relationship depth which the participants, who have taken this survey, develop while writing a program is roughly 6.

5.25 Do You Use the Property of Abstraction while Developing a Program?

In software engineering, deliberation is a system for overseeing unpredictability of PC frameworks. It lives up to expectations by creating a level of intricacy on which an individual collaborates with the framework, stifling more intricate subtle elements underneath the present level. The developer works with an admired interface (generally all around characterized), and can include extra levels of usefulness that would somehow be so mind boggling it that couldn’t be possible to handle (Abstraction (computer science), n.d.).

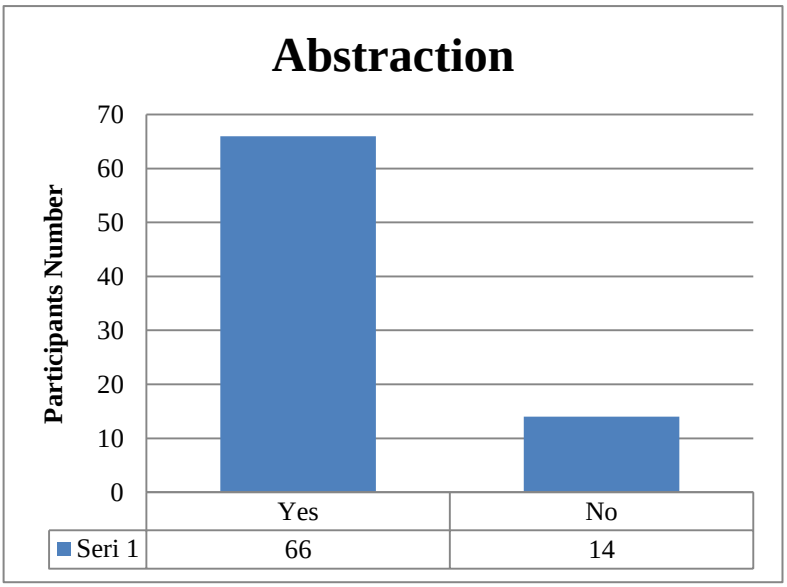

Figure 5.25 Abstraction

Based on the "Figure 5.25", among the participants who have taken the survey, about 81.5% of them use the property of abstraction, but 17% of them don't use this property while developing a program. In addition, 1 of them - the rest of the participants - sometimes uses, but sometimes doesn't use this property based on the projects while writing a program. Also, the diagram shows that most of the participants, who have taken this survey, use the property of abstraction while developing a program.

5.26 Do You Use the Property of Testability while Developing a Program?

Programming testability is the extent to which a product relic (i.e., a product framework, programming module, prerequisites or configuration archive) backings testing in a given test setting. On the off chance that the testability of the product antique is high, then discovering blames in the framework (on the off chance that it has any) by method for testing is simpler. Testability is not a characteristic property of a product antique, and can't be measured specifically (for example, programming size). Rather testability is an extraneous property which comes about because of interdependency of the product to be tried and the test objectives, test routines utilized and test assets (i.e., the test connection). A lower level of testability results in

expanded test exertion. In great cases, an absence of testability may thwart testing parts of the product or programming prerequisites by any stretch of the imagination.

To connection the testability with the trouble to discover potential blames in a framework (in the event that they exist) by testing it, a pertinent measure to survey the testability is how many numbers of experiments are required for every situation to shape a complete test suite (i.e., a test suite such that, in the wake of applying every test cases to the framework, gathered yields will give us a chance to unambiguously figure out if the framework is right or not as per some determination). On the off chance that this size is little, then the testability is high. Taking into account this measure, a testability chain of importance has been proposed (Software testability, n.d.).

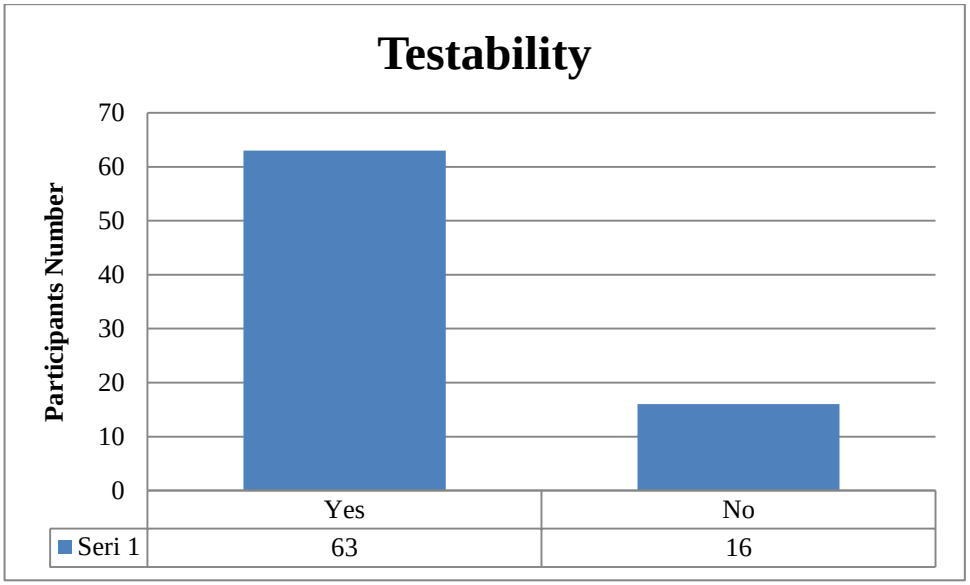

Figure 5.26 Testability

Based on the “Figure 5.26”, among the participants who have taken the survey, about 78% of them use the property of testability; however, 19.5% of them don’t use this property while developing a program. In addition, 2 of them - the rest of the participants - sometimes use, but sometimes don’t use this property based on the projects while writing a program. Moreover, the diagram shows that most of the participants, who have taken this survey, use the property of testability while developing a program.

5.27 Do You Use the Property of Use of Layers while Developing a Program?

In PC programming, layering is the association of programming into independent practical parts that associate in some consecutive and various leveled route, with every layer normally having an interface just to the layer above it and the layer underneath it (What is layering?, n.d.).

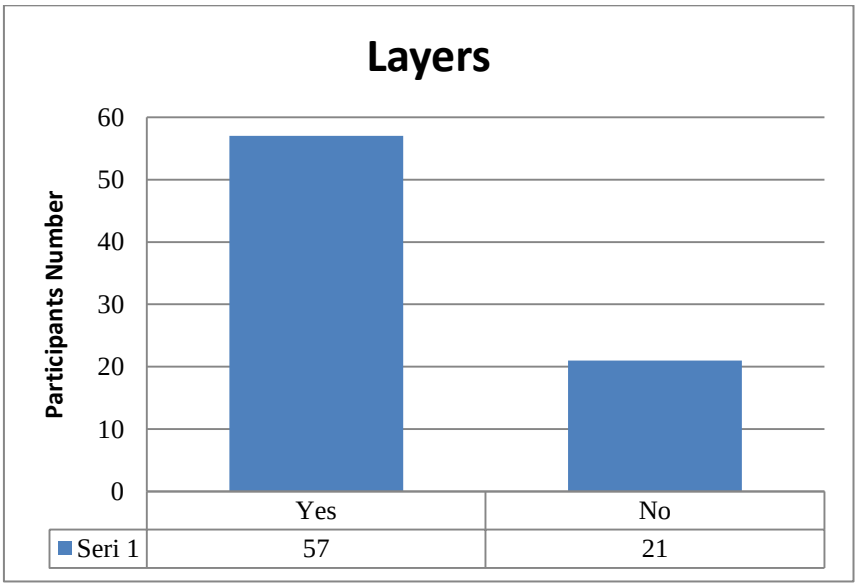

Figure 5.27 Layers

Based on the “Figure 5.27”, among the participants who have taken the survey, about 70% of them use the property of layers; however, 26% of them don’t use this property while developing a program. In addition, 3 of them - the rest of the participants - sometimes use, but sometimes don’t use this property based on the projects while writing a program.

5.28 Do You Use the Property of Layouts while Developing a Program?

In processing, design is the methodology of ascertaining the position of items in space subject to different requirements. This usefulness can be a piece of an application or bundled as a reusable segment or library. Page design is the calculation of the position of the sections, tabs, sentences, words and letters of content. This is finished by desktop distributed programming, typesetting programming and web program motors. These, thus, incorporate text style format and rendering motors that figure the right position of glyphs, which can be a test with complex scripts. Pictures can likewise be installed in the content. Another manifestation of design is found in format directors. They are a piece of gadget tool compartments, and can naturally ascertain a gadget's position in the light of arrangement limitations without the requirement for the developer to indicate total directions (Layout (computing), n.d.).

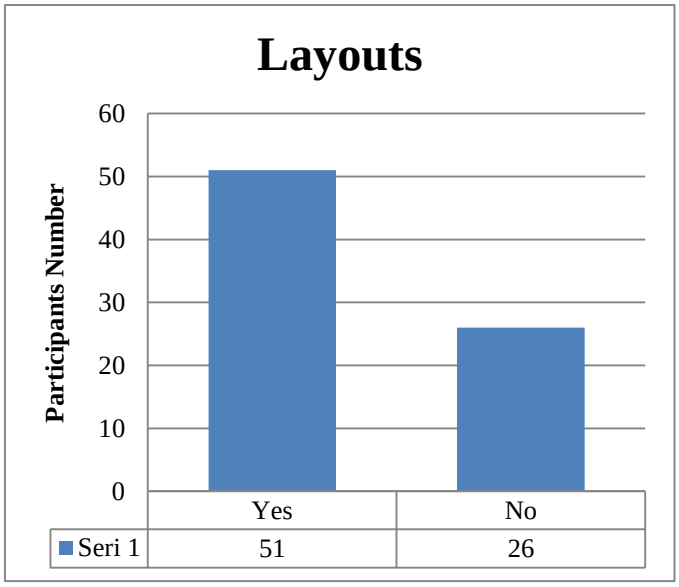

Figure 5.28 Layouts

Based on the "Figure 5.28", among the participants who have taken the survey, about 63% of them use the property of layouts; however, 32% of them don't use this property while developing a program. In addition, 4 of them - the rest of the participants - sometimes use, but sometimes don't use this property based on the projects while writing a program.

5.29 Do You Use the Property of Modularity while Developing a Program?

In programming configuration, measured quality alludes to a coherent dividing of the product plan that permits complex programming to be reasonable with the end goal of usage and upkeep. The rationale of dividing may be in the light of related capacities, usage contemplations, information joins or other criteria (Modularity, n.d.).

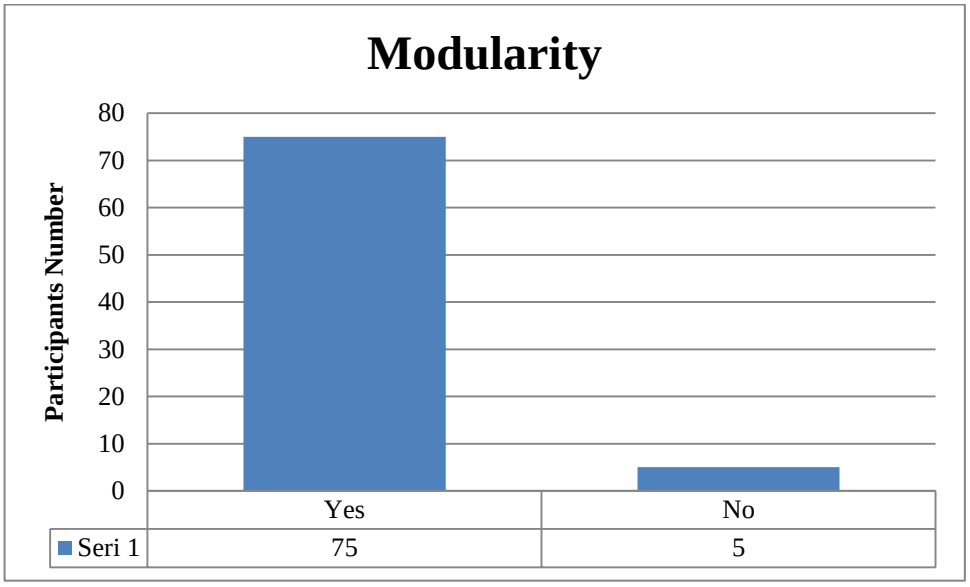

Figure 5.29 Modularity

Based on the “Figure 5.29”, among the participants who have taken the survey, about 92.5% of them use the property of modularity; however, 6% of them don’t use this property while developing a program. In addition, 1 of them - the rest of the participants - sometimes uses, but sometimes doesn’t use this property based on the projects while writing a program. Moreover, the diagram shows that almost all of the participants, who have taken this survey, use the property of modularity while developing a program.

5.30 Do You Use the Property of Coupling while Developing a Program?

In programming building, coupling is the way and level of relationship between programming modules, a measure of how nearly associated two schedules or modules are, the quality of the connections between modules. Coupling is normally appeared differently in relation to attachment. Low coupling frequently connects with high union and the other way around. Low coupling is frequently an indication of an all-around organized PC framework and a decent plan, and when joined with high attachment, underpins the general objectives of high coherence and viability (Coupling (computer programming), n.d.).

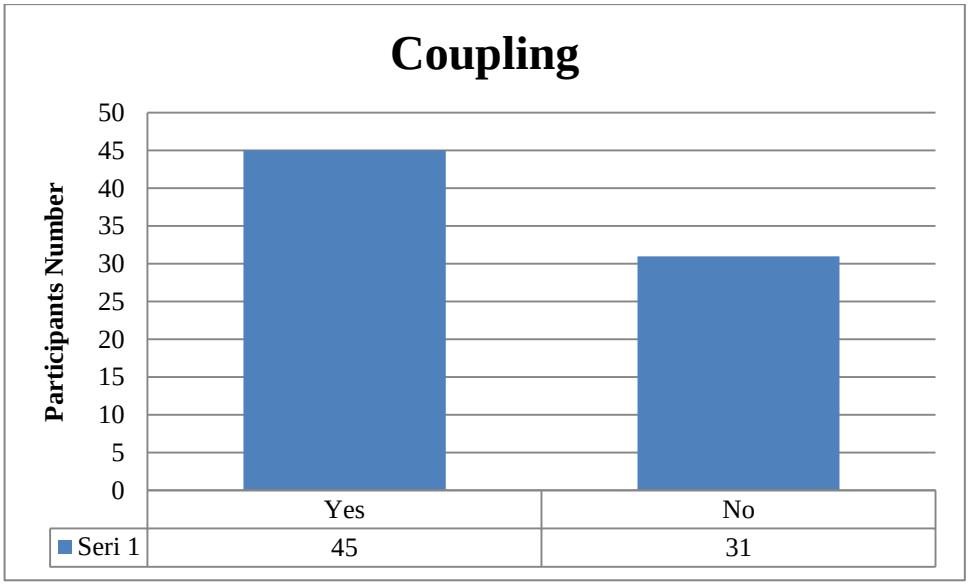

Figure 5.30 Coupling

Based on the "Figure 5.30", among the participants who have taken the survey, about 55.5% of them use the property of coupling; however, 38% of them don't use this property while developing a program. In addition, 5 of them - the rest of the participants - sometimes use, but sometimes don't use this property based on the projects while writing a program.

5.31 Do You Use the Property of Test Coverage while Developing a Program?

That is a measure of the extent of a project practiced by a test suite, generally communicated as a rate. This will normally include gathering data about which parts of a system are really executed when running the test suite so as to distinguish which branches of contingent explanations which have been taken (Test coverage in technology, n.d.).

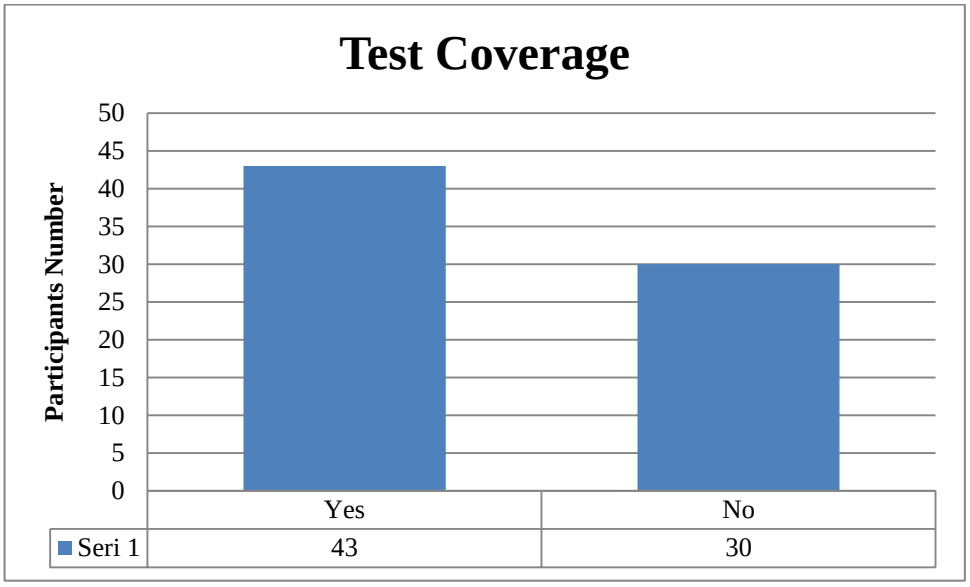

Figure 5.31 Test coverage

Based on the “Figure 5.31”, among the participants who have taken the survey, about 53% of them use the property of test coverage; however, 37% of them don’t use this property while developing a program. In addition, 8 of them - the rest of the participants - sometimes use, but sometimes don’t use this property based on the projects while writing a program.

5.32 Do You Use the Property of Error Handling while Developing a Program?

That is blunder taking care of alludes to the foresight, recognition and determination of programming, application and interchanges slips. Specific projects, called mistake handlers, are accessible for a few applications. The best projects of this sort thwart blunders if conceivable, recoup from them when they happen without ending the application or (as a last resort) nimbly end an influenced application and recovery the slip data to a log record (Error handling definition, n.d.).

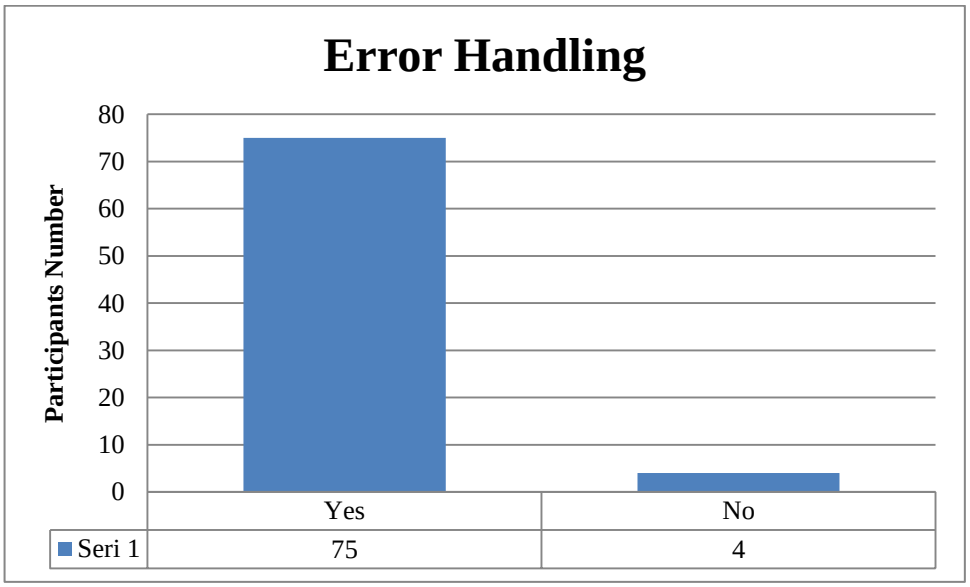

Figure 5.32 Error handling

Based on the "Figure 5.32", among the participants who have taken the survey, about 92.5% of them use the property of error handling; however, 5% of them don't use this property while developing a program. In addition, 2 of them - the rest of the participants - sometimes use, but sometimes don't use this property based on the projects while writing a program. Also, the diagram shows that almost all of the participants, who have taken this survey, use the property of error handling while developing a program.

5.33 Do You Use the Property of Exception Handling while Developing a Program?

Exemption taking care of is the procedure of reacting to the event, amid calculation, of special cases - bizarre or extraordinary conditions obliging exceptional transforming - frequently changing the ordinary stream of project execution. It is given by specific programming dialect builds or PC equipment instruments. By and large, a special case is taken care of (determined) by sparing the current condition of execution in a predefined place and changing the execution to a particular subroutine known as an exemption handler. On the off chance that exemptions are continual, the handler might later resume the execution at the first area utilizing the spared data. Case in point, a gliding point isolate by zero exemption will regularly, as a matter of course, permit the project to be continued while an out of memory condition may not be resolvable straightforwardly (Exception handling, n.d.).

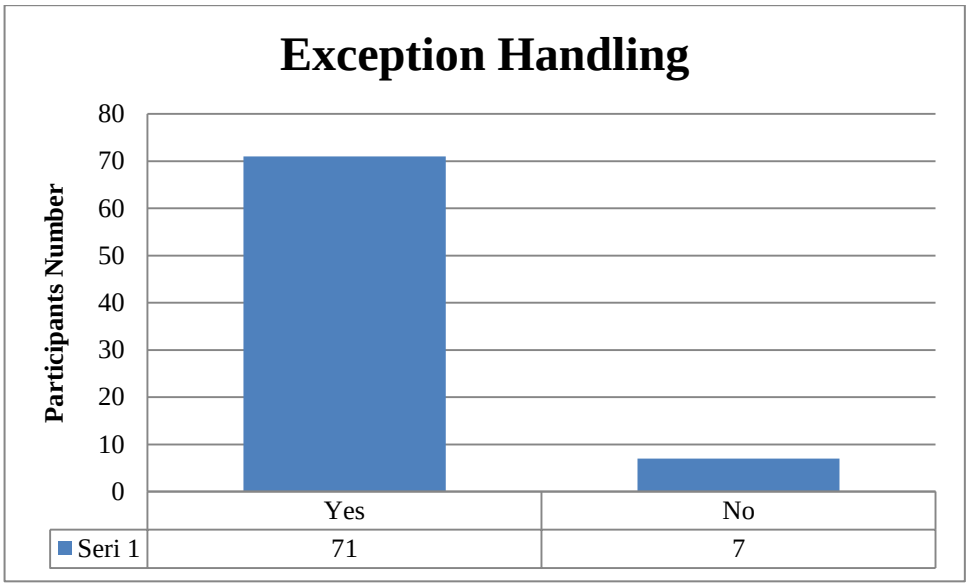

Figure 5.33 Exception handling

Based on the "Figure 5.33", among the participants who have taken the survey, about 87.5% of them use the property of exception handling; however, 8.5% of them don't use this property while developing a program. In addition, 3 of them - the rest of the participants - sometimes use, but sometimes don't use this property based on

the projects while writing a program. Furthermore, the diagram shows that most of the participants, who have taken this survey, use the property of exception handling while developing a program.

5.34 Do You Use the Property of Reusability while Developing a Program?

In software engineering and programming designing, reusability is the utilization of existing resources in some structure inside the product item advancement process. More than simple code, resources are items and by-results of the product improvement life cycle and incorporate programming parts, test suites, plans and documentation. Influence is changing existing resources as expected to meet particular framework prerequisites. Since reuse suggests making of an independently kept up rendition of the advantages, it is favored over leverage. Subroutines or capacities are the least difficult type of reuse. A lump of code is consistently composed utilizing modules or namespaces into layers. Defenders guarantee that questions and programming segments offer a more propelled type of reusability, in spite of the fact that it has been hard to unbiasedly gauge and characterize levels or scores of reusability. The capacity to reuse depends in a vital route on the capacity to manufacture bigger things from smaller parts and having the capacity to distinguish shared characteristics among those parts. Reusability is frequently an obliged normal for stage programming. Reusability conveys a few perspectives to programming advancement that don’t have to be considered when reusability is not needed. Reusability suggests some unequivocal administration of manufacture, bundling, appropriation, establishment, setup, arrangement, upkeep and overhaul issues. On the off chance that these issues are not considered, programming may give off an impression of being reusable from configuration perspective; however, it won’t be reused practically speaking. Programming reusability alludes to plan highlights of a product component that improves its suitability for reuse (Reusability, n.d.).

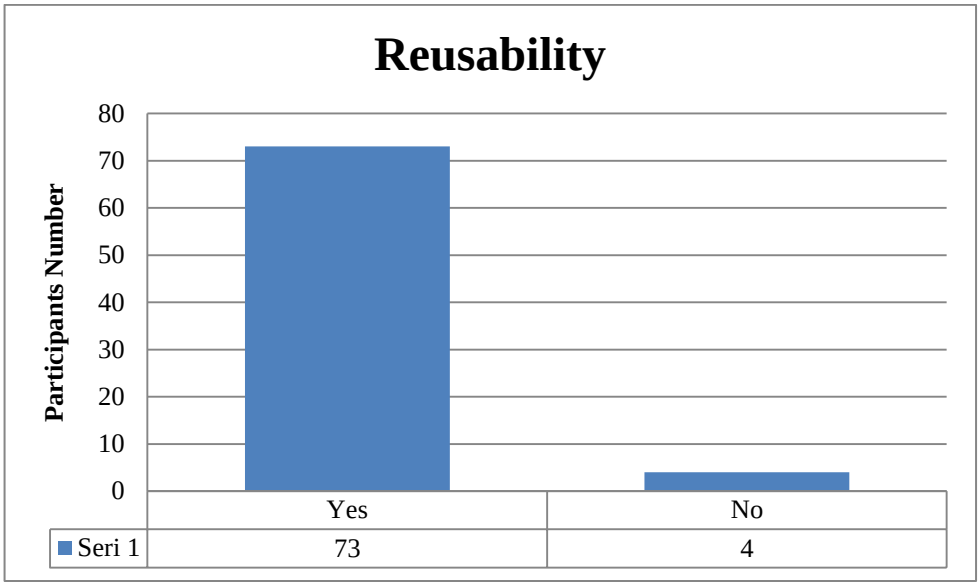

Figure 5.34 Reusability

Based on the “Figure 5.34”, among the participants who have taken the survey, about 90% of them use the property of reusability; however 5% of them don’t use this property while developing a program. In addition, 4 of them - the rest of the participants - sometimes use, but sometimes don’t use this property based on the projects while writing a program. Also, the diagram shows that almost all of the participants, who have taken this survey, use the property of reusability while developing a program.

5.35 How many Hours do You Work while Developing a Program for a Day?

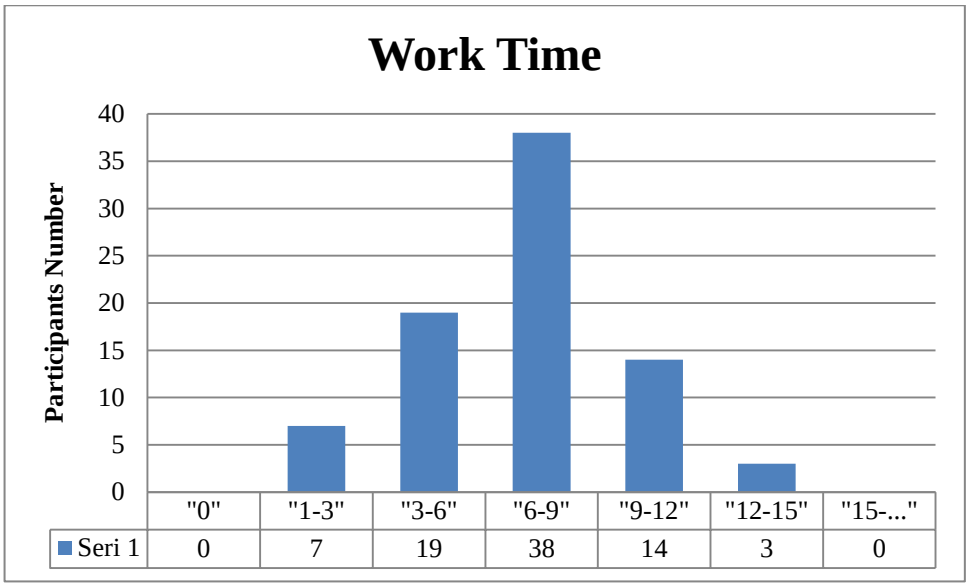

Figure 5.35 Work time

Based on the “Figure 5.35”, among the participants who have taken the survey, about 8.5% of them work for 1 to 3 hours, about 23.5% of them work for 3 to 6 hours, about 47% of them work for 6 to 9 hours, about 17% of them work for 9 to 12 hours, and about 4% of them work for 12 to 15 hours for a day on average while developing a program. In addition, the diagram shows that all participants work every day, but none of them works for more than 15 hours on average in a day while writing a program. Moreover, the diagram shows that most of the participants, who have taken this survey, work for less than 9 hours on average in a day while developing a program. Also, the average working time which the participants, who have taken the survey, have while writing a program is roughly 7 hours for a day.

5.36 What is the Average Time of Thinking about a Program before Developing it?

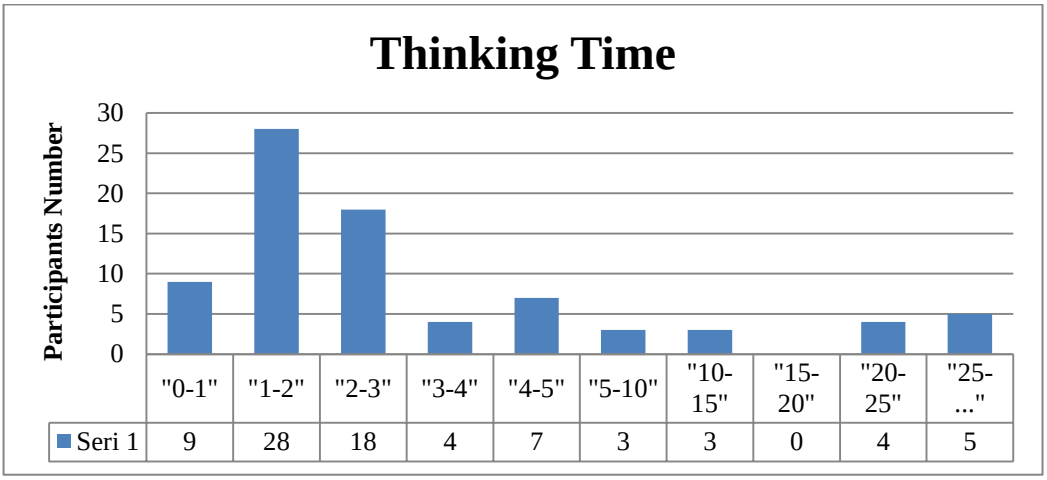

Figure 5.36 Thinking time

Based on the "Figure 5.36", among the participants who have taken the survey, about 11% of them think about a program for less than 1 hour, about 34.5% of them think about a program for 1 to 2 hours, about 22% of them think about a program for 2 to 3 hours, about 5% of them think about a program for 3 to 4 hours, about 8.5% of them think about a program for 4 to 5 hours, about 4% of them think about a program for 5 to 10 hours, about 4% of them think about a program for 10 to 15 hours, about 5% of them think about a program for 20 to 25 hours, and about 6% of them think about a program for more than 25 hours on average before developing it. In addition, the diagram shows that none of the participants, which have taken this survey, has chosen the thinking time groups 15 to 20 hours in this question. Moreover, the diagram shows that most of the participants think about a program for less than 5 hours on average before writing it. Also, the average thinking time which the participants, who have taken the survey, have about a program before developing it is roughly 6 hours.

5.37 What is the Average Time of the Completion of Developing a Program?

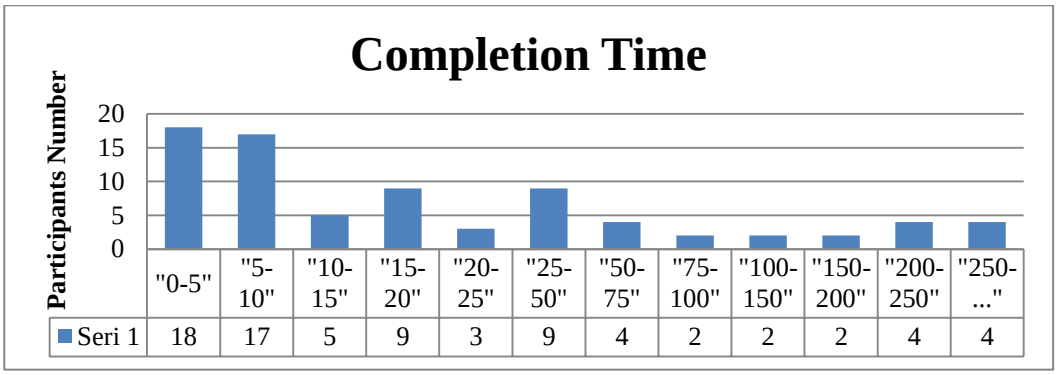

Figure 5.37 Completion time

Based on the “Figure 5.37”, among the participants who have taken the survey, about 22% of them complete a program in less than 5 hours, about 21% of them complete a program in 5 to 10 hours, about 6% of them complete a program in 10 to 15 hours, about 11% of them complete a program in 15 to 20 hours, about 3.5% of them complete a program in 20 to 25 hours, about 11% of them complete a program in 25 to 50 hours, about 5% of them complete a program in 50 to 75 hours, about 2.5% of them complete a program in 75 to 100 hours, about 2.5% of them complete a program in 100 to 150 hours, about 2.5% of them complete a program in 150 to 200 hours, about 5% of them complete a program in 200 to 250 hours, and about 5% of them complete a program in more than 250 hours on average while developing it. In addition, two of them - the rest of the participants - haven’t answered this question based on the given options by writing “changes based on the projects” in the survey. Furthermore, the diagram shows that most of the participants, who have taken this survey, complete to develop a program in less than 75 hours on average. Also, the average completion time of writing a program is roughly 50 hours among the participants, who have taken the survey.

5.38 What is the Average Time of the Revision of a Program in General?

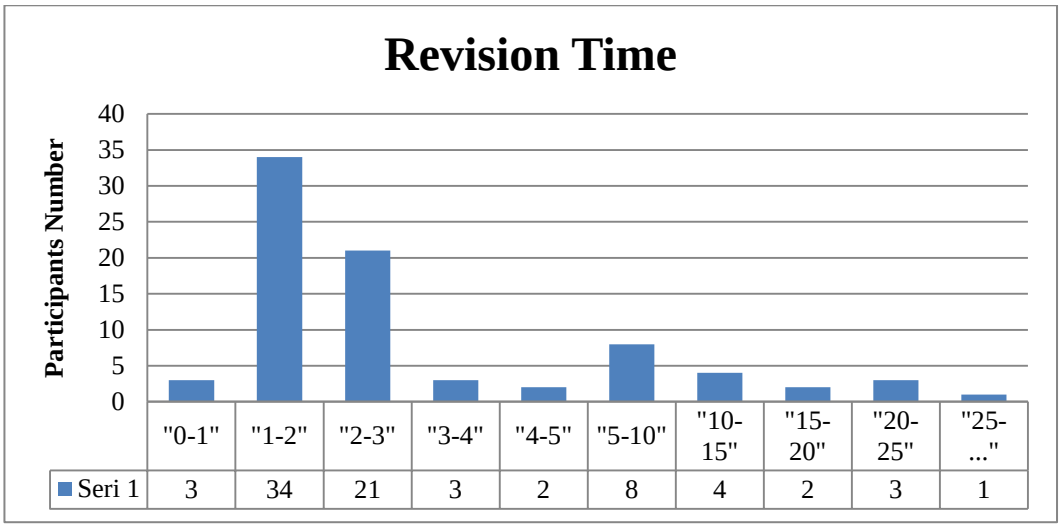

Figure 5.38 Revision time

Based on the “Figure 5.38”, among the participants who have taken the survey, about 3.5% of them review a program in less than 1 hour, about 42% of them review a program in 1 to 2 hours, about 26% of them review a program in 2 to 3 hours, about 3.5% of them review a program in 3 to 4 hours, about 2.5% of them review a program in 4 to 5 hours, about 10% of them review a program in 5 to 10 hours, about 5% of them review a program in 10 to 15 hours, about 2.5% of them review a program in 15 to 20 hours, about 3.5% of them review a program in 20 to 25 hours, and about 1.5% of them review a program in more than 25 hours on average after developing it. Also, the diagram shows that most participants, who have taken this survey, review a program in less than 10 hours on average after writing it. In addition, the average revision time of a program after developing it is roughly 4 hours among the participants, who have taken the survey.

5.39 What is the Average Time of the Early Delivery of the Programs?

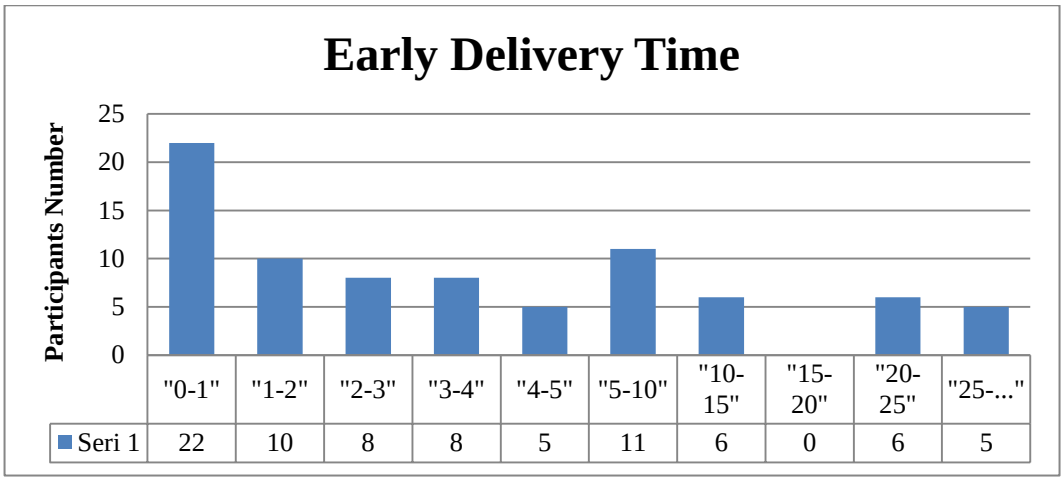

Figure 5.39 Early delivery time

Based on the “Figure 5.39”, among the participants who have taken the survey, about 27% of them deliver a program earlier in less than 1 hour, about 12.5% of them deliver a program earlier in 1 to 2 hours, about 10% of them deliver a program earlier in 2 to 3 hours, about 10% of them deliver a program earlier in 3 to 4 hours, about 6% of them deliver a program earlier in 4 to 5 hours, about 13.5% of them deliver a program earlier in 5 to 10 hours, about 7.5% of them deliver a program earlier in 10 to 15 hours, about 7.5% of them deliver a program earlier in 20 to 25 hours, and about 6% of them deliver a program earlier in more than 25 hours on average after developing it. Also, the diagram shows that none of the participants has chosen time groups 15 to 20 hours in this question in the survey. Moreover, the diagram shows that most of the participants, who have taken this survey, deliver a program earlier in less than 10 hours on average after writing it. In addition, the average early delivery time of a program after developing it is roughly 9 hours among the participants, who have taken the survey.

5.40 What is the Average Time of the Delay of the Delivery of the Programs?

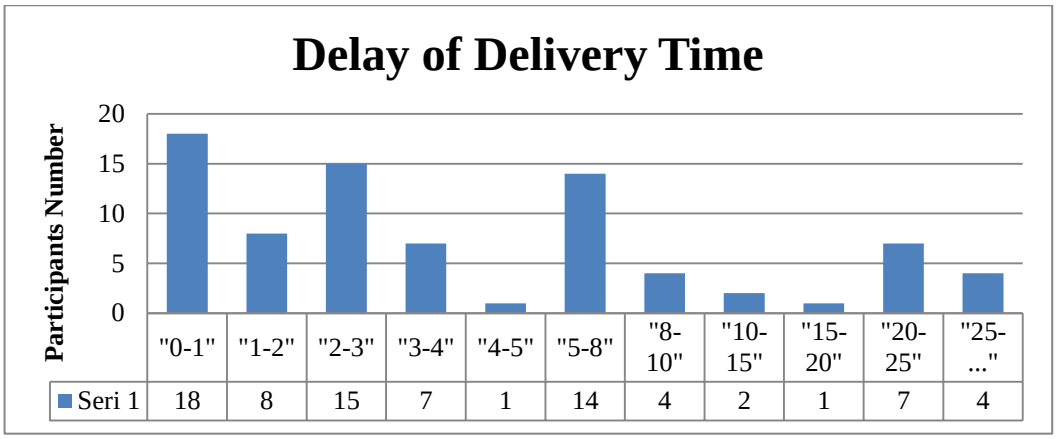

Figure 5.40 Delay of delivery time

Based on the “Figure 5.40”, among the participants who have taken the survey, about 22% of them deliver a program latter in less than 1 hour, about 10% of them deliver a program latter in 1 to 2 hours, about 18.5% of them deliver a program latter in 2 to 3 hours, about 8.5% of them deliver a program latter in 3 to 4 hours, about 1.5% of them deliver a program latter in 4 to 5 hours, about 17% of them deliver a program latter in 5 to 8 hours, about 5% of them deliver a program latter in 8 to 10 hours, about 2.5% of them deliver a program latter in 10 to 15 hours, about 1.5% of them deliver a program latter in 15 to 20 hours, about 8.5% of them deliver a program latter in 20 to 25 hours, and about 5% of them deliver a program latter in more than 25 hours on average after developing it. Also, the diagram shows that most of the participants, who have taken this survey, deliver a program in less than 8 hours on average after writing it. In addition, the average delay of delivery time of a program after developing it is roughly 7 hours among the participants, who have taken the survey.

5.41 What is the Average Timing Error in General?

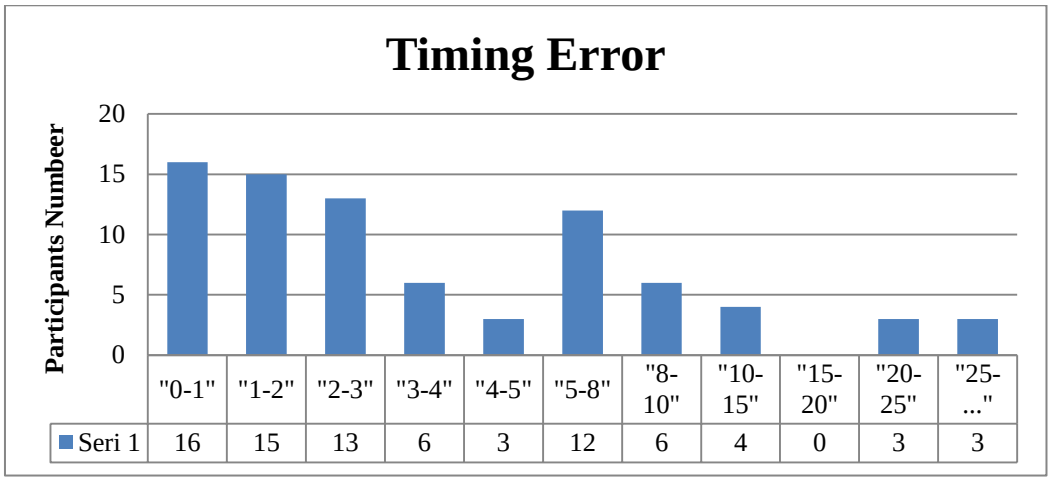

Figure 5.41 Timing error

Based on the “Figure 5.41”, among the participants who have taken the survey, about 20% of them have less than 1 hour timing error, about 18.5% of them have 1 to 2 hours timing error, about 16% of them have 2 to 3 hours timing error, about 7.5% of them have 3 to 4 hours timing error, about 3.5% of them have 4 to 5 hours timing error, about 15% of them have 5 to 8 hours timing error, about 7.5% of them have 8 to 10 hours timing error, about 5% of them have 10 to 15 hours timing error, about 3.5% of them have 20 to 25 hours timing error, and about 3.5% of them have more than 25 hours timing error on average while developing a program. Also, the diagram shows that none of the participants has chosen the timing error groups 15 to 20 hours in this question in the survey. Moreover, the diagram shows that most of the participants, who have taken this survey, have less than 8 hours timing error on average while writing a program. In addition, the average timing error while developing a program is roughly 5 hours among the participants, who have taken the survey.

5.42 What is the Average Size Error in General?

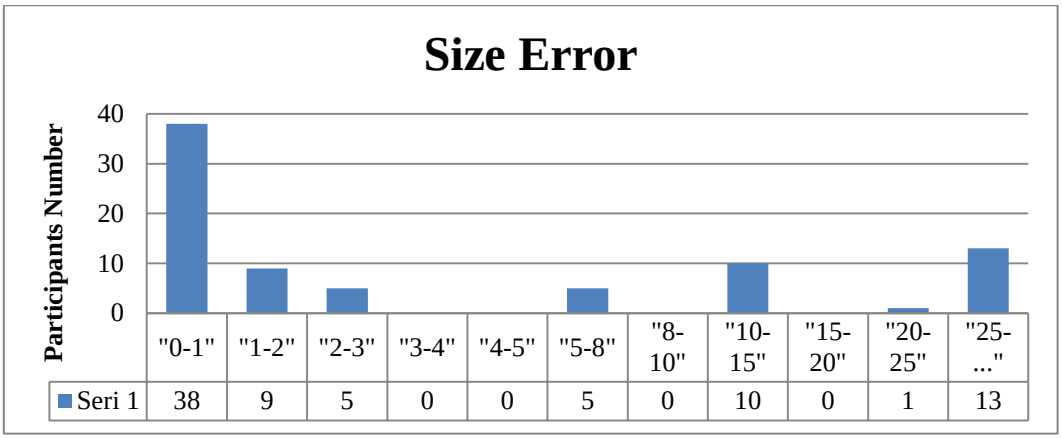

Figure 5.42 Size error

Based on the “Figure 5.42”, among the participants who have taken the survey, about 47% of them have less than 1 MB size error, about 11% of them have 1 to 2 MBs size error, about 6% of them have 2 to 3 MBs size error, about 6% of them have 5 to 8 MBs size error, about 12.5% of them have 10 to 15 MBs size error, about 1.5% of them have 20 to 25 MBs size error, and about 16% of them have more than 25 MBs size error on average while developing a program. Also, the diagram shows that none of the participants has chosen the size error groups 3 to 4 MBs, 4 to 5 MBs, 8 to 10 MBs and 15 to 20 MBs in this question in the survey. In addition, the average size error while writing a program is roughly 30 MBs among the participants, who have taken the survey.

5.43 Coded Undetermined Features

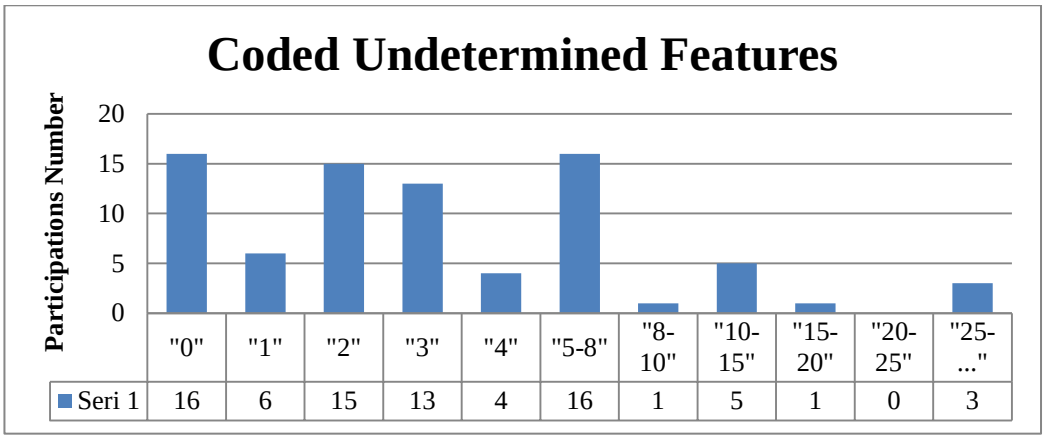

Figure 5.43 Coded undetermined features

Based on the “Figure 5.43”, among the participants who have taken the survey, about 19.5% of them don’t code any undetermined features, about 7.5% of them code only undetermined 1 feature, about 18.5% of them code undetermined 2 features, about 16% of them code undetermined 3 features, about 5% of them code undetermined 4 features, about 19.5% of them code undetermined 5 to 8 features, about 1.5% of them code undetermined 8 to 10 features, about 6% of them code undetermined 10 to 15 features, about 1.5% of them code undetermined 15 to 20 features, and about 3.5% of them code undetermined more than 25 features on average in general while developing a program. Also, the diagram shows that none of the participants has chosen the groups undetermined 20 to 25 features in this question in the survey. Moreover, 1 of them - the rest of the participants - hasn’t answered this question based on the options by writing “doing stress tests” in the survey. Furthermore, the diagram shows that most of the participants, who have taken this survey, code undetermined less than 8 features on average in general while writing a program. In addition, the average features number, which the project doesn’t determine at the analysis stage in general, is roughly 7 among the participants, who have taken the survey, while developing a program.

5.44 Segmentation Error

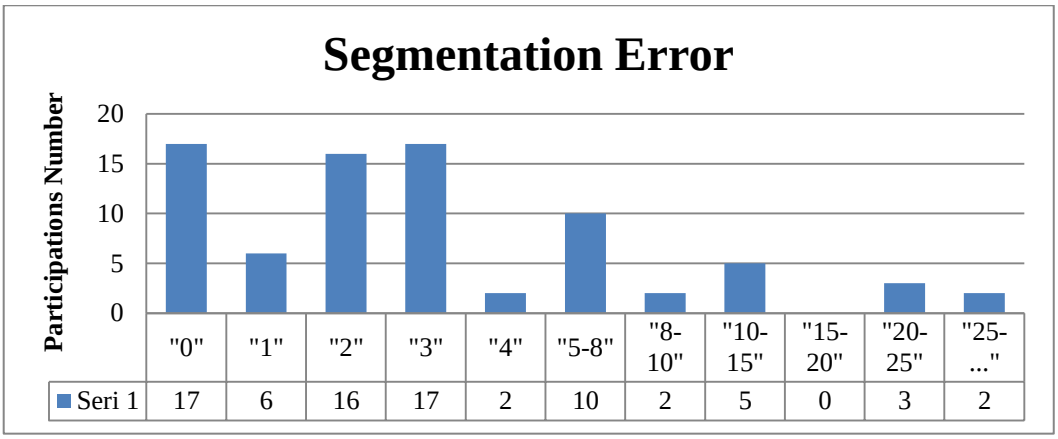

Figure 5.44 Segmentation error

Based on the “Figure 5.44”, among the participants who have taken the survey, about 21% of them don’t have any segmentation errors, about 7.5% of them have only 1 segmentation error, about 19.5% of them have 2 segmentation errors, about 21% of them have 3 segmentation errors, about 2.5% of them have 4 segmentation errors, about 12% of them have 5 to 8 segmentation errors, about 2.5% of them have 8 to 10 segmentation errors, about 6% of them have 10 to 15 segmentation errors, about 4% of them have 20 to 25 segmentation errors, and about 2.5% of them have more than 25 segmentation errors on average in general while developing a program. Also, the diagram shows that none of the participants has chosen the segmentation error groups 15 to 20 in this question in the survey. Moreover, 1 of them - the rest of the participants - hasn’t answered this question based on the options by writing “there is some difference” in the survey. Furthermore, the diagram shows that most of the participants, who have taken this survey, have less than 8 segmentation errors on average in general while writing a program. In addition, the average segmentation error number in general is roughly 4 among the participants, who have taken the survey, while developing a program.

5.45 Meeting Outputs

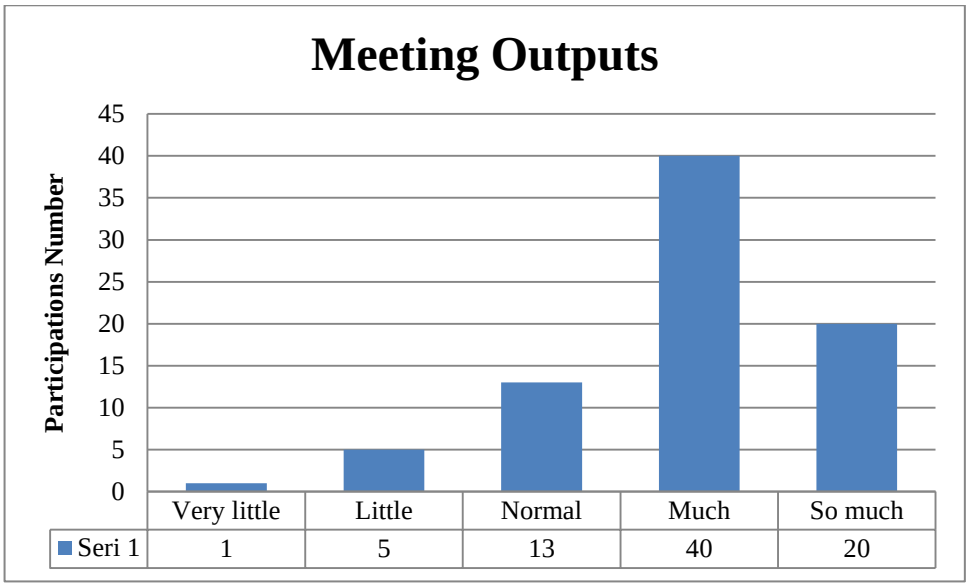

Figure 5.45 Meeting outputs

Based on the "Figure 5.45", among the participants who have taken the survey, about 1% of them meet the outputs very little, about 6% of them meet the outputs little, about 16% of them meet the outputs normal, about 49.5% of them meet the outputs much, and about 24.5% of them meet the outputs which are determined at the analysis stage so much while developing a program. In addition, 2 of them - the rest of the participants - meet the outputs sometimes much, but sometimes too much based on the projects while writing a program. Also, the diagram shows that most of the participants, who have taken this survey, meet the outputs which are determined at the analysis stage more than normal while developing a program.

5.46 Do You Think that the Program Makes Your Customers' Works Easy?

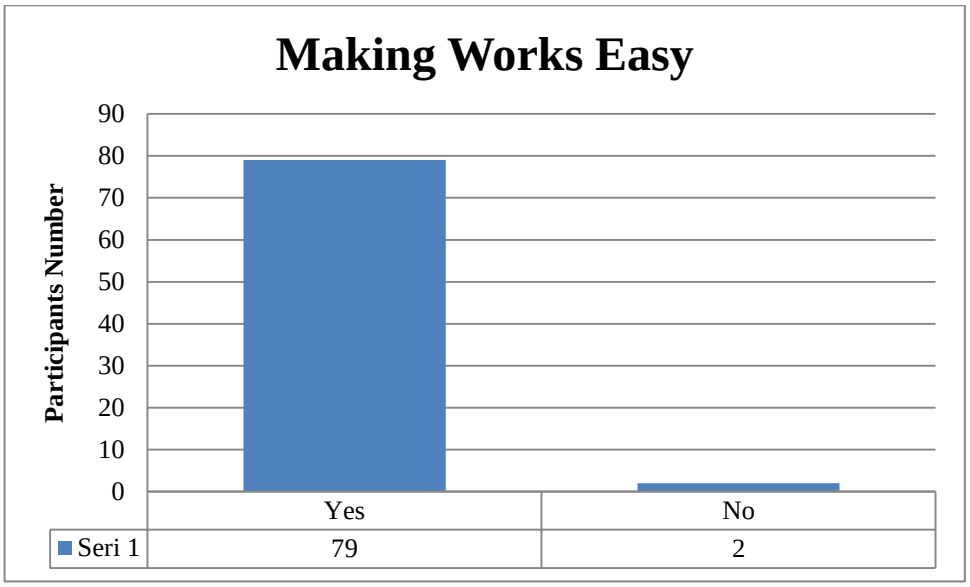

Figure 5.46 Making works easy

Based on the "Figure 5.46", among the participants who have taken the survey, about 97.5% of them think that the program which they have developed makes the customers' works easy; however, about 2.5% of them don't think so. Also, the diagram shows that almost all of the participants, who have taken this survey, think that their developed programs make the customers' works easy.

5.47 Do You Think that You have Developed a Practical Program?

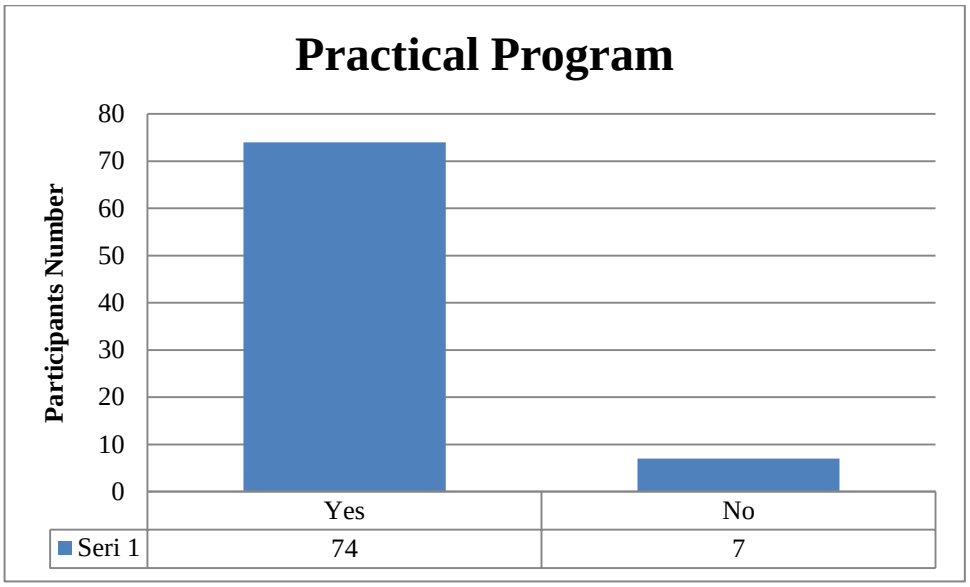

Figure 5.47 Practical program

Based on the "Figure 5.47", among the participants who have taken the survey, about 91.5% of them think that they have developed a practical - usable and understandable - program; however, about 8.5% of them don't think so. In addition, the diagram shows that most of the participants, who have taken this survey, think that they have written a usable and understandable program.

5.48 As Company, do You Guarantee the Quality of Your Software?

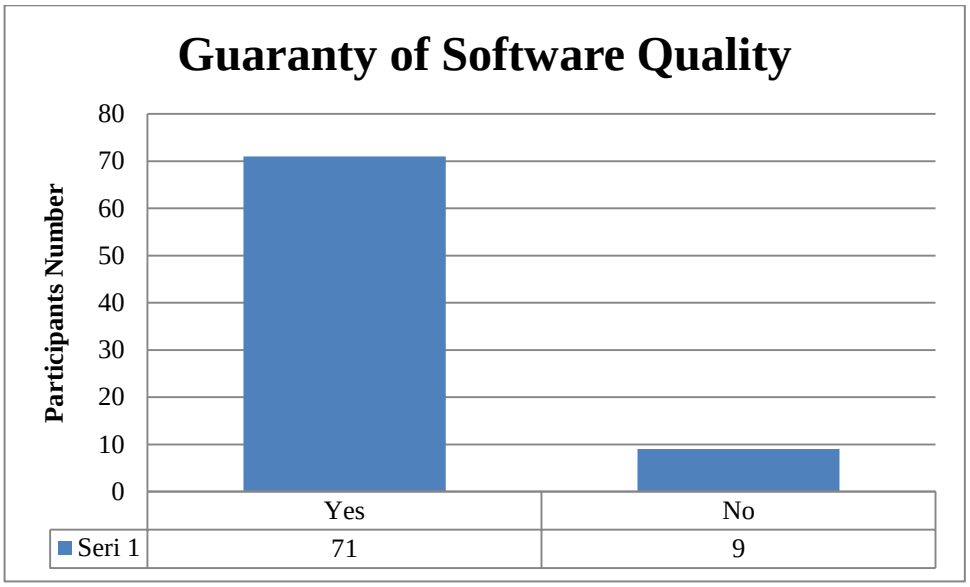

Figure 5.48 Guaranty of software quality

Based on the "Figure 5.48", among the participants who have taken the survey, about 87.5% of them guarantee the quality of their software as company; however, about 11% of them don't do that. In addition, 1 of them - the rest of the participants - sometimes gives guaranty of their software as company, but sometimes doesn't do that based on the projects. Moreover, the diagram shows that most of the participants, who have taken this survey, guarantee their software quality as company.

5.49 Individually, do You Pay Attention to the Quality in the Program?

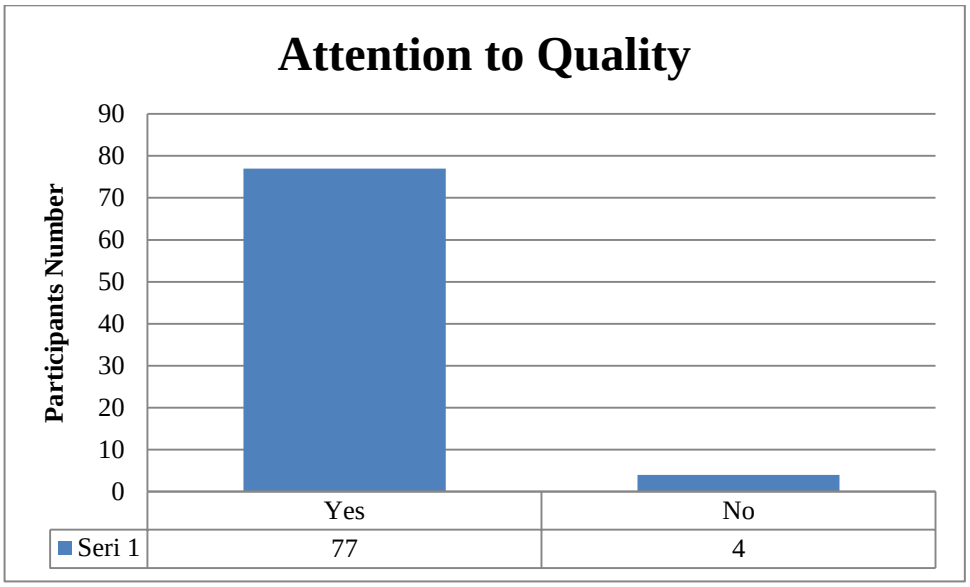

Figure 5.49 Attention to quality

Based on the "Figure 5.49", among the participants who have taken the survey, about 95% of them pay attention to the quality in the program which they have developed; however, about 5% of them don't do that. Furthermore, the diagram shows that almost all of the participants, who have taken this survey, pay attention to the quality in the program, which they have written.

5.50 What is Your Opinion on the Quality of the Program?

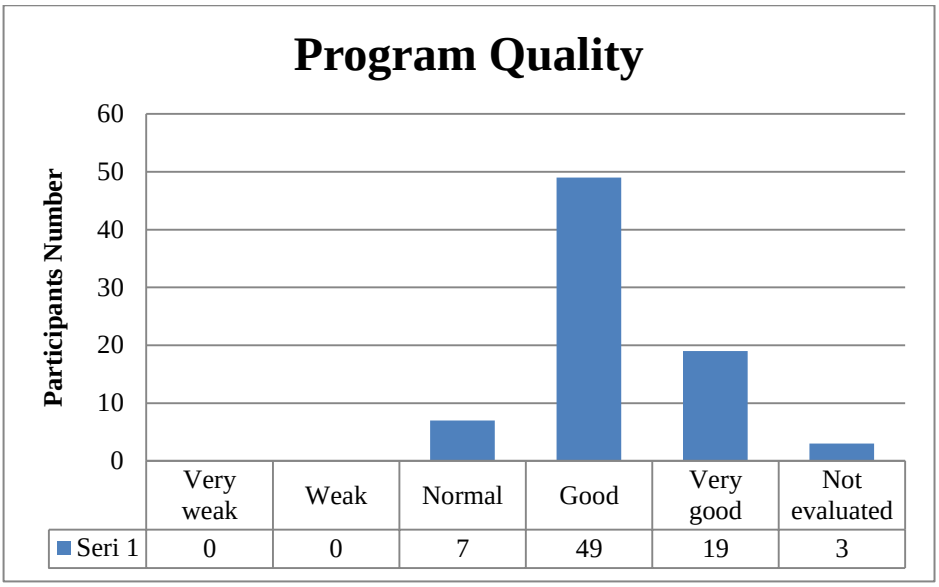

Figure 5.50 Program quality

Based on the "Figure 5.50", among the participants who have taken the survey, about 8.5% of them think that the quality of the program is normal, about 60.5% of them think that the quality of the program is good, about 23.5% of them think that the quality of the program is very good, and about 3.5% of them do not care the quality of the program which they have developed. In addition, 3 of them - the rest of the participants - think that the quality of the program is sometimes good, but sometimes very good based on the projects. Moreover, the diagram shows that most of the participants, who have taken this survey, think that the quality of the program, which they have written, is more than normal.

5.51 What is Your Degree of the Attention to the Reliable Code Writing?

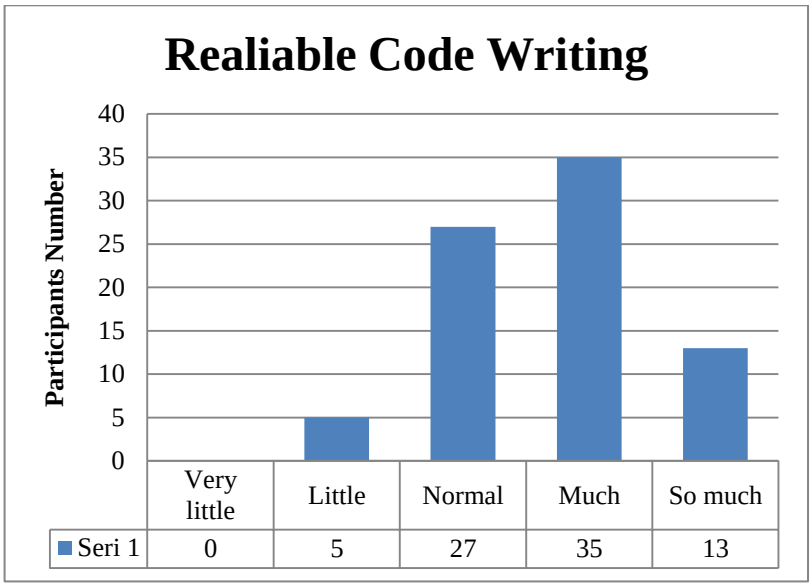

Figure 5.51 Reliable code writing

Based on the “Figure 5.51”, among the participants who have taken the survey, about 6% of them pay attention to the reliable code writing little, about 33.5% of them pay attention to the reliable code writing normal, about 43% of them pay attention to the reliable code writing much, and about 16% of them pay attention to the reliable code writing so much while developing a program. In addition, 1 of them - the rest of the participants - pays attention to the reliable code writing sometimes much, but sometimes too much based on the projects while writing a program. Also, the diagram shows that most of the participants, who have taken this survey, pay attention to the reliable code writing more than little while developing a program.

5.52 Individually, do You Give Tech Support to the Customers after Release?

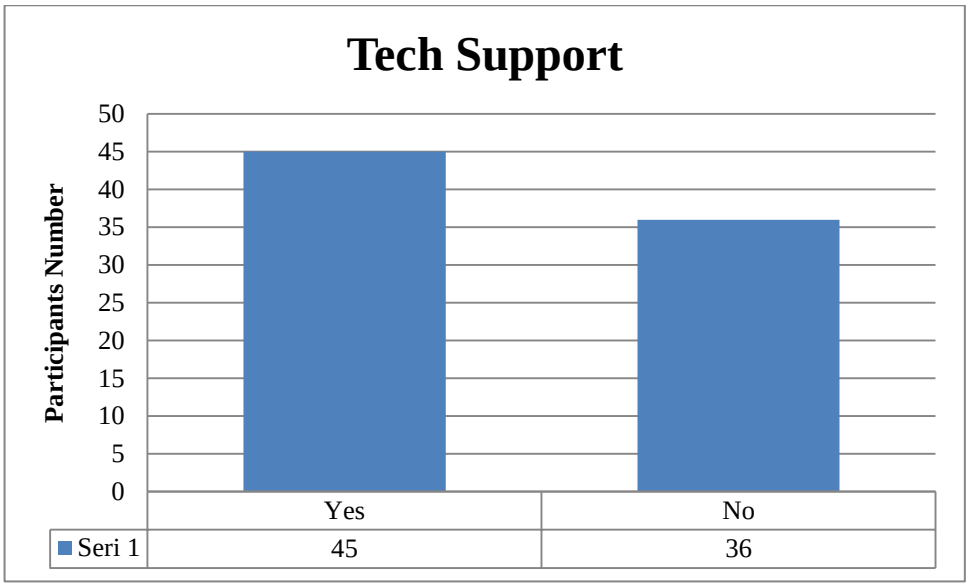

Figure 5.52 Tech support

Based on the "Figure 5.52", among the participants who have taken the survey, about 55.5% of them give tech support to the customers individually after the program, which they have developed, has been completed (delivered to the customers); however, about 44.5% of them don't do that individually.

5.53 Project Revision

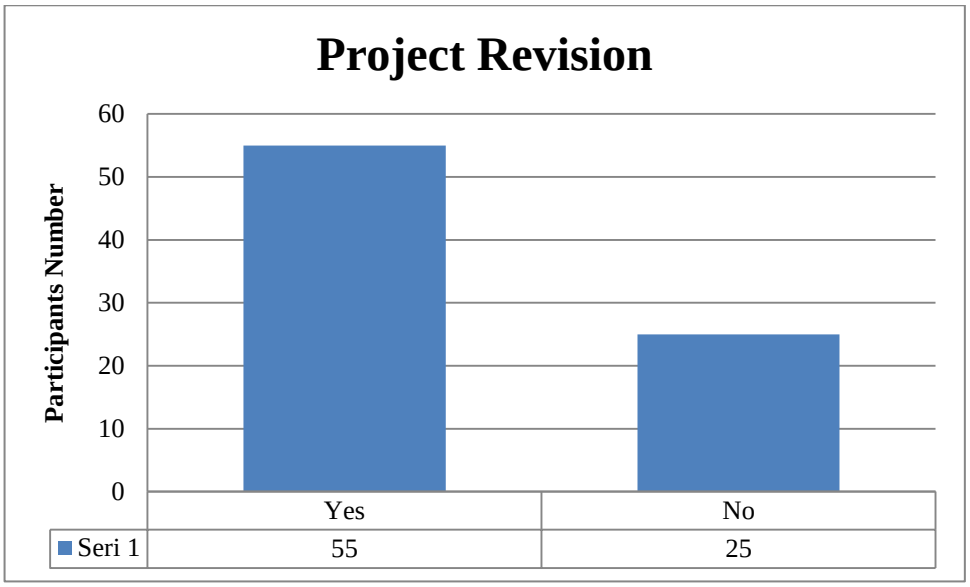

Figure 5.53 Project revision

Based on the "Figure 5.53", among the participants who have taken the survey, about 68% of them review the project, which they have developed, after it is delivered to the customers due to the obligation of their software development process; however, about 30.5% of them don't have a software development process which has an obligation like that. In addition, 1 of them - the rest of the participants - sometimes does this revision, but sometimes doesn't do that because of the change of her/his software development process.

5.54 Program Difficulty

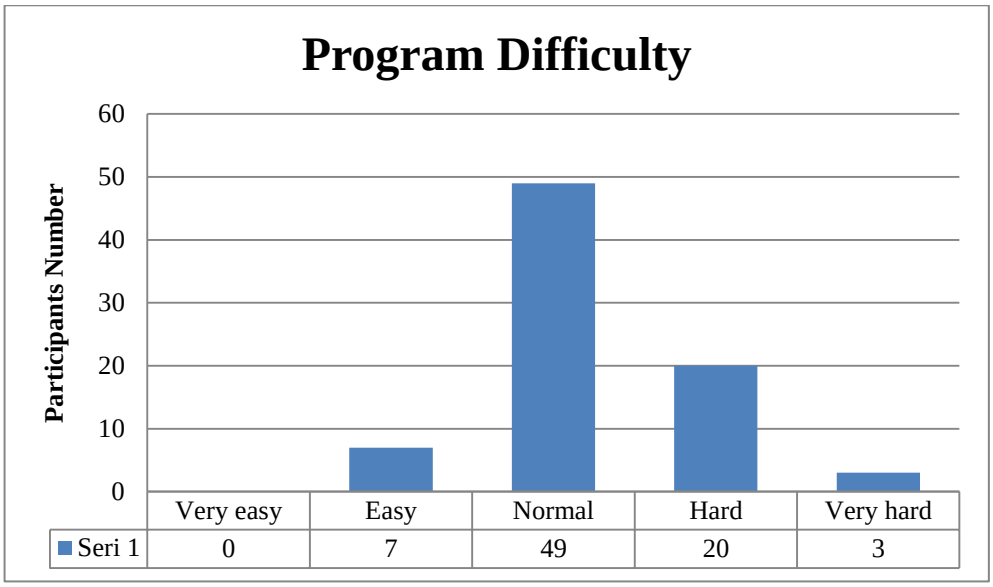

Figure 5.54 Program difficulty

Based on the "Figure 5.54", among the participants who have taken the survey, about 8.5% of them think that their developing program is easy to write, about 60.5% of them think that their developing program is normal to write, about 24.5% of them think that their developing program is hard to write, and about 4% of them think that their developing program is very hard to write. In addition, 1 of the participants thinks that their developing program is sometimes easy to write, but sometimes normal to write, and also, 1 of the participants thinks that their developing program is sometimes hard to write, but sometimes very hard to write based on the projects. Furthermore, the diagram shows that most of the participants, who have taken this survey, think that their developing program is harder than easy to write.

5.55 Do You Work Alone or in a Team while Programming?

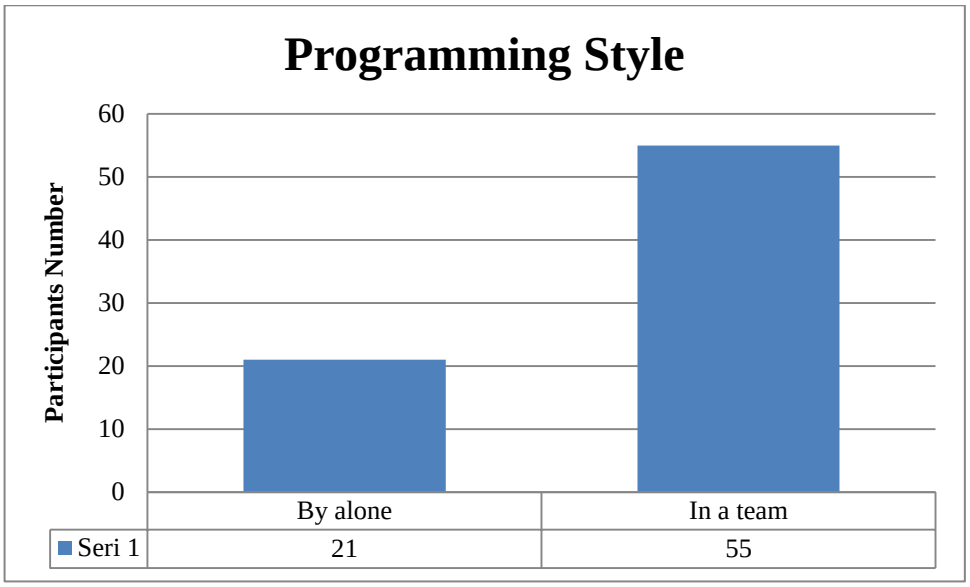

Figure 5.55 Programming style

Based on the “Figure 5.55”, among the participants who have taken the survey, about 26% of them work alone; however, about 68% of them work in a team while programming. In addition, 5 of them - the rest of the participants - sometimes work alone, but sometimes work in a team while programming based on the projects.

5.56 Do You Do Pair Programming?

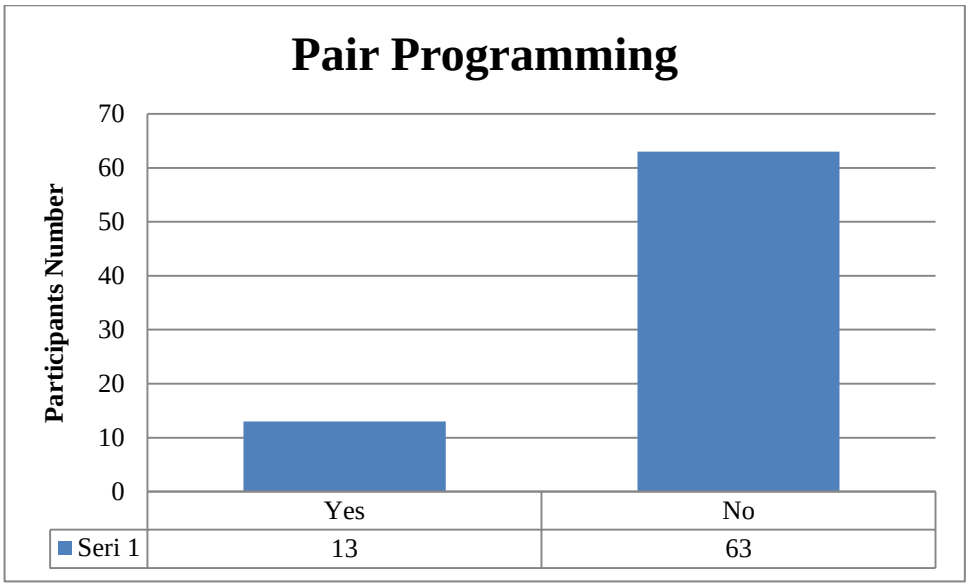

Figure 5.56 Pair programming

Based on the "Figure 5.56", among the participants who have taken the survey, about 16% of them work in pairs; however, about 78% of them don't work in pairs while programming. In addition, 5 of them - the rest of the participants - sometimes work in pairs, but sometimes work alone while programming based on the projects. Also, the diagram shows that most of the participants, who have taken this survey, don't work in pairs while developing a program.

5.57 Do You Have an Opportunity to Use Your Creativity while Programming?

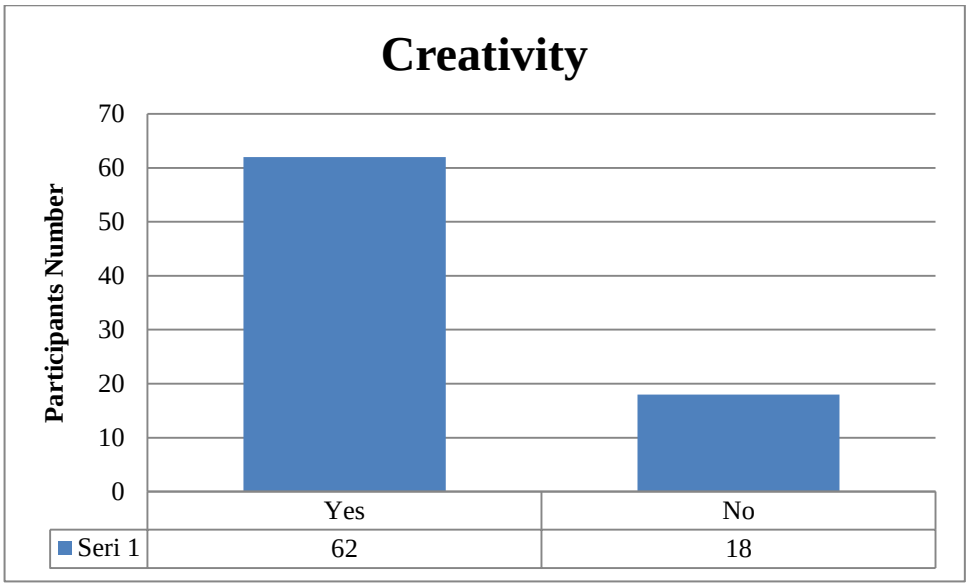

Figure 5.57 Creativity

Based on the "Figure 5.57", among the participants who have taken the survey, about 76.5% of them have an opportunity to use their creativity while programming; however, about 22% of them don't have that. In addition, 1 of them - the rest of the participants - sometimes has this opportunity, but sometimes doesn't have that based on the projects. Moreover, the diagram shows that most of the participants, who have taken this survey, have an opportunity to use their creativity while developing a program.

5.58 Are You able to Be Original while Writing (Developing) a Program?

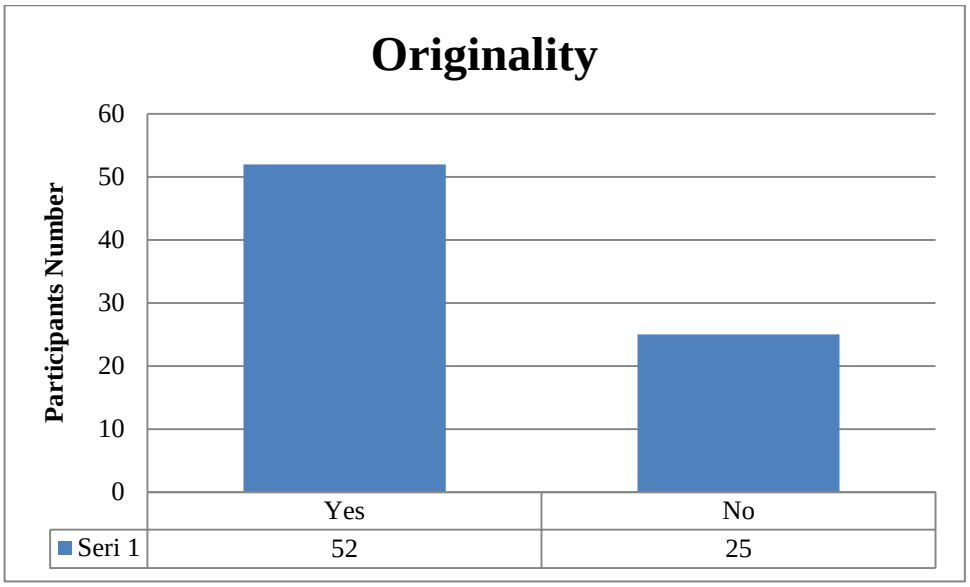

Figure 5.58 Originality

Based on the "Figure 5.58", among the participants who have taken the survey, about 64% of them are able to be original while developing a program; however, about 31% of them aren't. In addition, 4 of them - the rest of the participants - are able to be sometimes original, but sometimes aren't based on the projects while programming.

5.59 Change of Software Development Tool

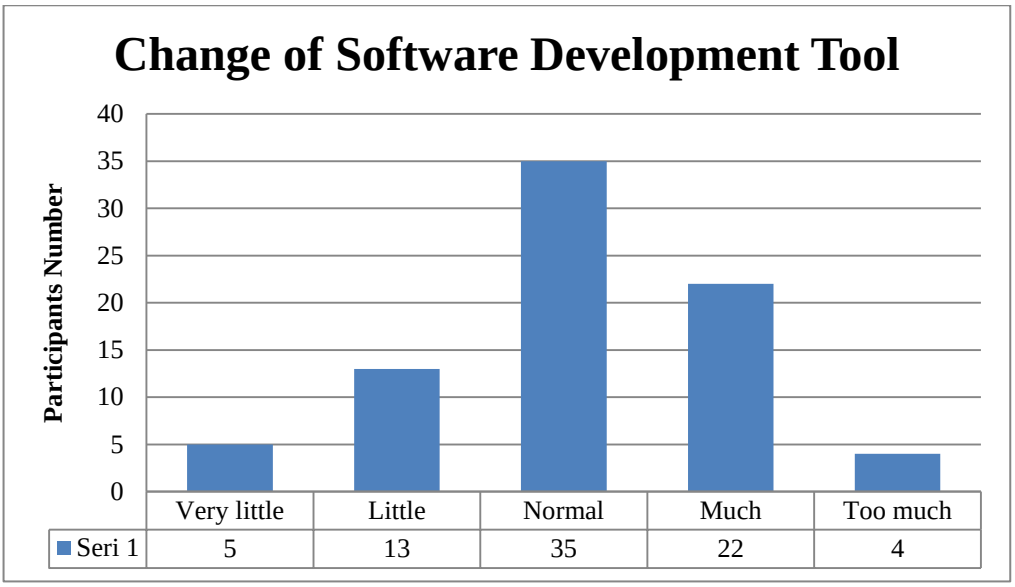

Figure 5.59 Change of software development tool

Based on the "Figure 5.59", among the participants who have taken the survey, about 6% of them think that the effect, which the software development tool that has been used before is changed, to the program, which they have developed, would be very little, about 16% of them think that this effect would be little to the program, 43% of them think that this effect would be normal to the program, 27% of them think that this effect would be much to the program, and 5% of them think that this effect would be too much to the program. In addition, 1 of the participants thinks that the effect, which the software development tool that has been used before is changed, to the program, which they have developed, would be sometimes very little, but sometimes little based on the projects, and also, 1 of the participants hasn't commented about this question in the survey. Moreover, the diagram shows that most of the participants, who have taken this survey, think that the effect, which the software development tool that has been used before is changed, to the program, which they have developed, would be more than little.

5.60 Something Changed

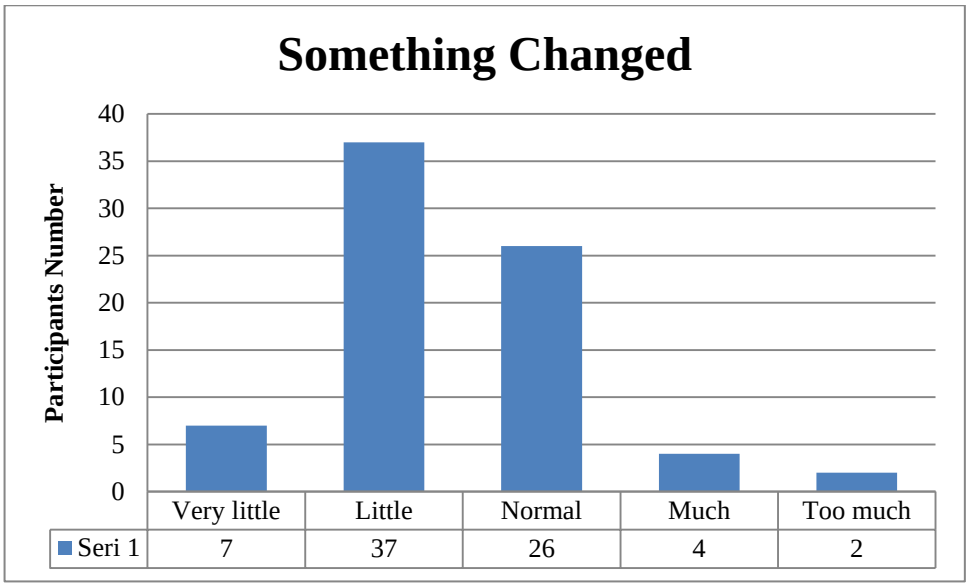

Figure 5.60 Something changed

Based on the "Figure 5.60", among the participants who have taken the survey, about 8.5% of them think that the effect to the program, which they have developed, in the case that if some changes are made in this program after it has been completed would be very little, about 45.5% of them think that this effect would be little to the program, 32% of them think that this effect would be normal to the program, 5% of them think that this effect would be much to the program, and 2.5% of them think that this effect would be too much to the program. In addition, 1 of the participants thinks that the effect to the program, which s/he has developed, in the case that if some changes are made in this program after it has been completed would be sometimes very little, but sometimes little, and also, 2 of the participants think that this effect would be sometimes little, but sometimes normal to the program based on the projects. Moreover, 1 of the participants thinks that this effect would be sometimes normal, but sometimes much to the program based on the projects. Furthermore, 1 of the participants hasn't commented about this question in the survey. Also, the diagram shows that most of the participants, who have taken this survey, think that the effect to the program, which they have developed, in the case that if some changes are made in this program after it has been completed would be less than much.

5.61 Do You Depend on the Mechanism in the Software Development Process?

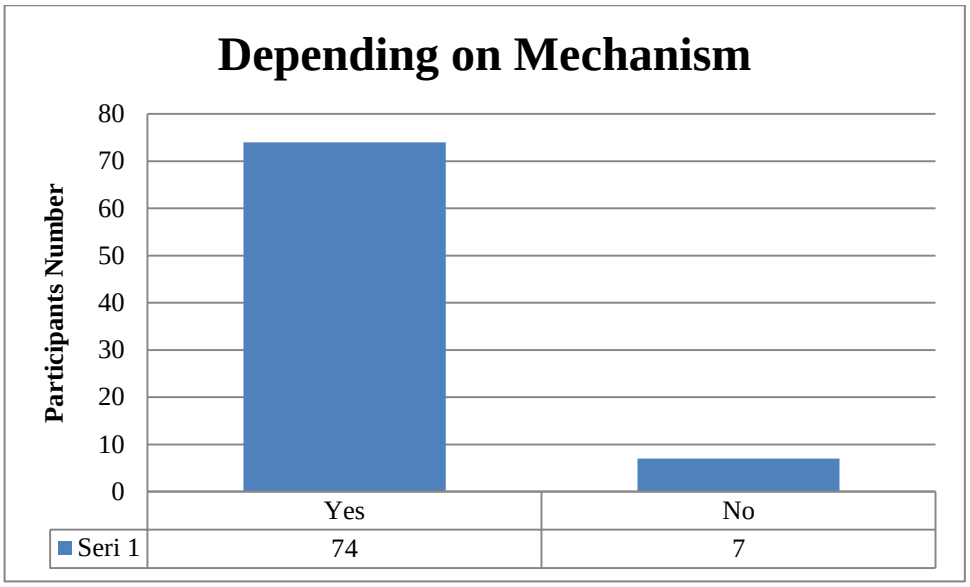

Figure 5.61 Depending on mechanism

Based on the "Figure 5.61", among the participants who have taken the survey, about 91.5% of them depend on the mechanism in the software development process while developing a program; however, about 8.5% of them don't do that. In addition, the diagram shows that most of the participants, who have taken this survey, depend on the mechanism in the software development process while programming.

5.62 Are You Asked for a Report about the Program which You have Developed?

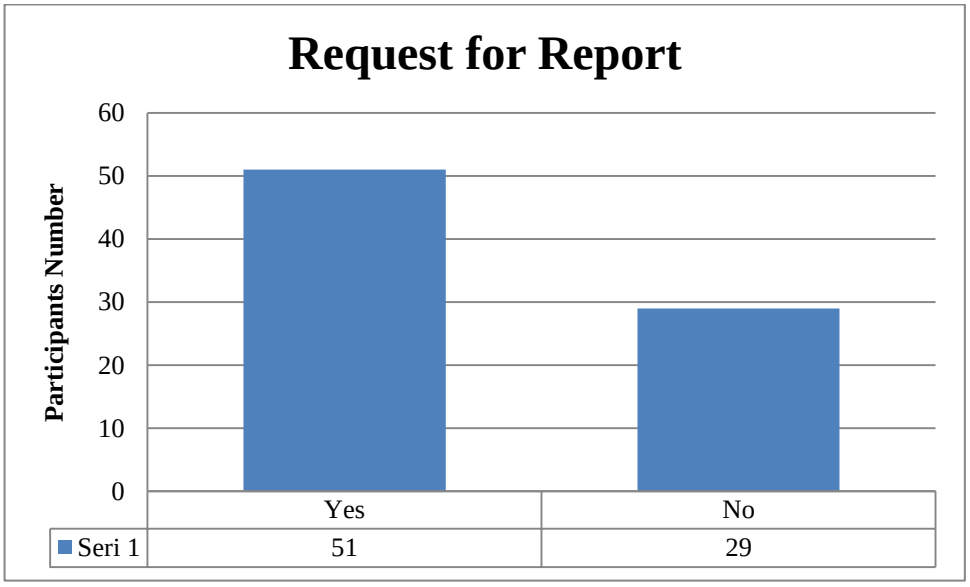

Figure 5.62 Request for report

Based on the "Figure 5.62", among the participants who have taken the survey, about 63% of them are asked for a report about the program which they have developed; however, about 35.5% of them aren't requested that. In addition, 1 of them - the rest of the participants - sometimes is requested this report from her/his, but sometimes isn't based on the projects.

5.63 Is There a Well-Defined Reporting System in the Company where You Work?

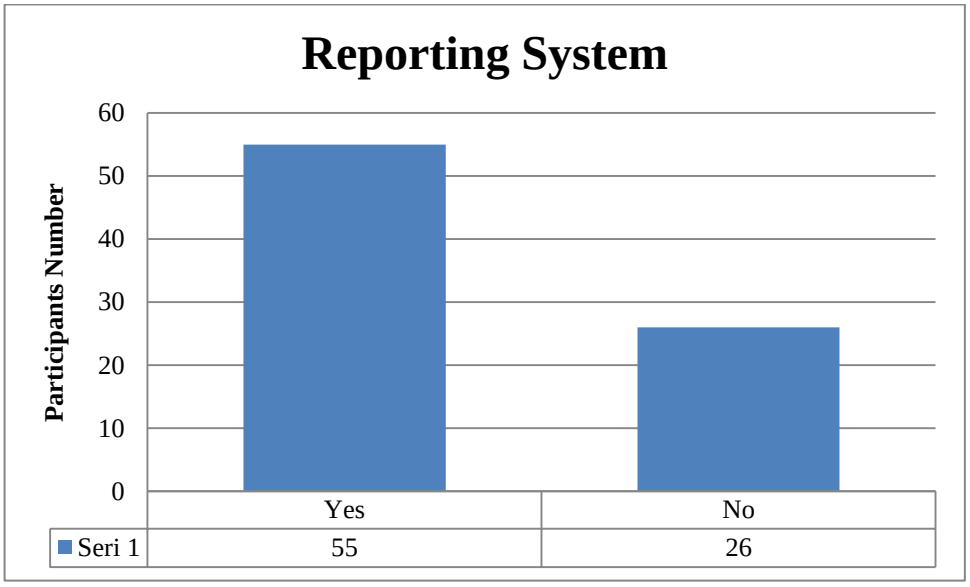

Figure 5.63 Reporting system

Based on the "Figure 5.63", among the participants who have taken the survey, about 68% of them have a well-defined reporting system in the company where they work; however, about 32% of them don't have that.

CHAPTER SIX
INFERENCE OF CRITERIA

6.1 The Algorithm of Inference - Association Rule Mining Algorithm - Apriori

Affiliation guideline era is normally part up into two different steps:

- To start with, least backing is connected to locate all regular item-sets in a database.
- Second, these regular item-sets and the base certainty imperative are utilized to shape rules.

While the second phase is true, the first phase needs more consideration. Discovering all successive item-sets in a database is troublesome since it includes looking all conceivable item-sets (thing mixes). The arrangement of conceivable item-sets is the force situated over *I* and has size $2^n - 1$ (barring the unfilled set which is not a legitimate item-set). Despite the fact that the measure of the power-set develops exponentially in the number of things *n* in *I*, proficient pursuit is conceivable utilizing the descending conclusion property of bolster (too called hostile to monotonicity) which ensures that for an incessant item-set, every one of its subsets are likewise visit and along these lines for a rare item-set, all its supersets should likewise be occasional. Misusing this property, productive calculations (e.g., Apriori and Eclat) can locate all successive item-sets.

```
procedure Apriori (T, minSupport) { //T is the database and minSupport is the minimum support
        L1= {frequent items};
        for (k= 2; Lk-1 !=∅; k++) {
                Ck= candidates generated from Lk-1
                //that iscartesian product Lk-1 x Lk-1 and eliminating any k-1 size itemset that is not
                //frequent
                for each transaction t in database do{
                        #increment the count of all candidates in Ck that are contained in t
                        Lk = candidates in Ck with minSupport
                }//end for each
        }//end for
        return ⋃k Lk;
}
```

Figure 6.1 Pseudo-code - the apriori algorithm

As it is regular in affiliation mining of rule, given an arrangement of item-sets (for example, sets of retail exchanges, every posting individual things acquired), the calculation endeavors to discover subsets which are basic to no less than a base number *C* of the item-sets. Apriori utilizes a "base up" methodology, where continuous subsets are augmented one thing at once (a stage known as hopeful era), and gatherings of applicants are tried against the information. The calculation ends when no further fruitful augmentations are found.

Apriori utilizes broadness first pursuit and a tree structure to tally hopeful thing sets proficiently. It creates competitor thing arrangements of length *k* from thing arrangements of length *k - 1*. At that point, it prunes the hopefuls which have an occasional sub design. As indicated by the descending conclusion lemma, the hopeful set contains all regular *k-length* thing sets. After that, it checks the exchange database to focus visit thing sets among the competitors.

Apriori, while verifiably critical, experiences various inefficiencies or exchange offs, which have brought forth different calculations. Competitor era creates huge quantities of subsets (the calculation endeavors to load up the applicant set with whatever number as could be allowed before every output). Base up subset investigation (basically a broadness first traversal of the subset grid) finds any maximal subset *S* strictly when each of the $2^{|S|} - 1$ of its fitting subsets (Mining frequent itemsets - apriori algorithm, n.d.).

6.2 Inference-1

Table 6.1 Inference-1

Question	Answer
C.1. Do you reuse any code parts while writing (developing) a program?	YES
C.4. Do you use any software development method while writing (developing) a program?	YES
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the “Table 6.1”, among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.3 Inference-2

Table 6.2 Inference-2

Question	Answer
C.1. Do you reuse any code parts while writing (developing) a program?	YES
C.4. Do you use any software development method while writing (developing) a program?	YES
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the “Table 6.2”, among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.4 Inference-3

Table 6.3 Inference-3

Question	Answer
C.4. Do you use any software development method while writing (developing) a program?	YES
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the “Table 6.3”, among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.5 Inference-4

Table 6.4 Inference-4

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.14. As company, do you guarantee the quality of your software?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.4", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.6 Inference-5

Table 6.5 Inference-5

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.14. As company, do you guarantee the quality of your software?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.5", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.7 Inference-6

Table 6.6 Inference-6

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.14. As company, do you guarantee the quality of your software?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.6", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.8 Inference-7

Table 6.7 Inference-7

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES
E.8. Do you depend on the mechanism in the software development process?	YES

Based on the "Table 6.7", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.9 Inference-8

Table 6.8 Inference-8

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the “Table 6.8”, among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.10 Inference-9

Table 6.9 Inference-9

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the “Table 6.9”, among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 77.

6.11 Inference-10

Table 6.10 Inference-10

Question	Answer
C.1. Do you reuse any code parts while writing (developing) a program?	YES
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.10", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.12 Inference-11

Table 6.11 Inference-11

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
C.10.10. Do you use the property of Reusability while writing (developing) a program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.11", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.13 Inference-12

Table 6.12 Inference-12

Question	Answer
C.6. Do you use any methods or models which have been determined before while writing (developing) a program?	YES
C.10.5. Do you use the property of Modularity while writing (developing) a program?	YES
C.10.8. Do you use the property of Error Handling while writing (developing) a program?	YES
D.13. Do you think that you have written (developed) a practical (usable&understandable) program?	YES
D.15. Individually, do you pay attention to the quality in the program which you have written (developed)?	YES

Based on the "Table 6.12", among the participants who have taken the survey, have answered these questions in the same direction with the percent of roundly 76.

6.14 The Algorithm of Trees - Decision Tree Algorithm - C4.5

C4.5 is accumulation of calculations for operating arrangements in mining of data and learning of machine. It adds to the characterization display as being a tree of decision. Also, by selecting a base case, it shows possible ways which reach this base case by displaying a decision tree. Thus, which ways go to the goal and what the goal depends on can be learned and understood easily. C4.5 comprises of three gatherings in this calculation: C4.5, C4.5-rules and C4.5-no-pruning. In this rundown, it is going to be concentrated on the fundamental C4.5 calculation. More or less, C4.5 is executed repeatedly with this taking after arrangement:

- Check if calculation fulfills end criteria.
- Compute data theoretic criteria for all properties.
- Pick best credit as indicated by the data theoretic criteria.
- Make a choice hub in view of the best quality in step 3.
- Affect the dataset in the light of recently made choice hub in step 4.
- For all sub-dataset in step 5, call C4.5 calculation to get a sub-tree (recursive call).

- Connect the tree got in step 6 to the choice hub in step 4.
- Return tree.

Input: an attribute-valued dataset D

1: Tree = {}
2: **if** D is "pure" OR other stopping criteria met **then**
3: terminate
4: **end if**
5: **for all** attribute $a \in D$ **do**
6: Compute information-theoretic criteria if we split on a
7: **end for**
8: a_{best} = Best attribute according to above computed criteria
9: Tree = Create a decision node that tests a_{best} in the root
10: D_v = Induced sub-datasets from D based on a_{best}
11: **for all** D_v **do**
12: $\text{Tree}_v = \text{C4.5}(D_v)$
13: Attach Tree_v to the corresponding branch of Tree
14: **end for**
15: **return** Tree

Figure 6.2 Pseudo-code of the algorithm - C4.5

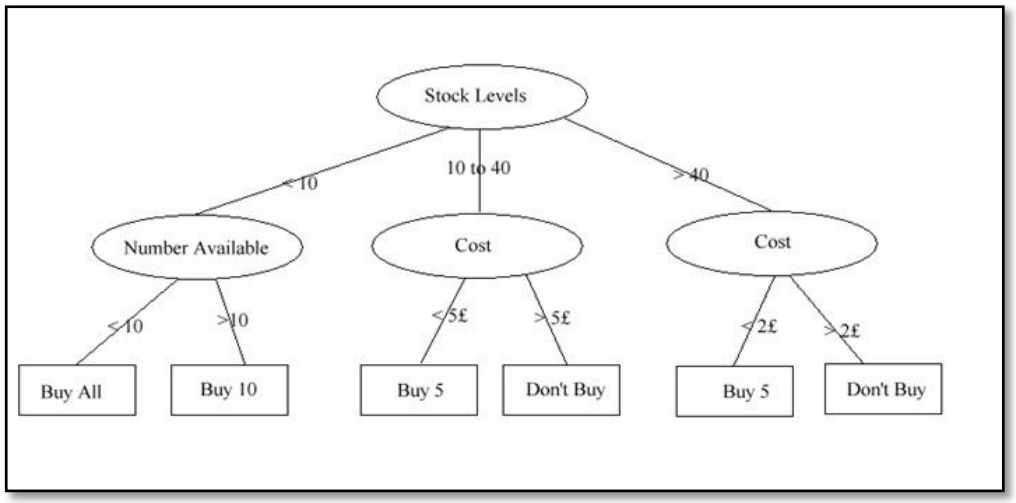

Figure 6.3 An example of a decision tree built by using the algorithm - C4.5

Algorithm-C4.5 is suitable for true issues in the world as it manages numeric and missing qualities. The calculation can be utilized for building less or bigger, more precise trees of decision, and the calculation is very time-productive (Golb, 2013).

6.15 The Relation of Developed Methods to Software Development Method

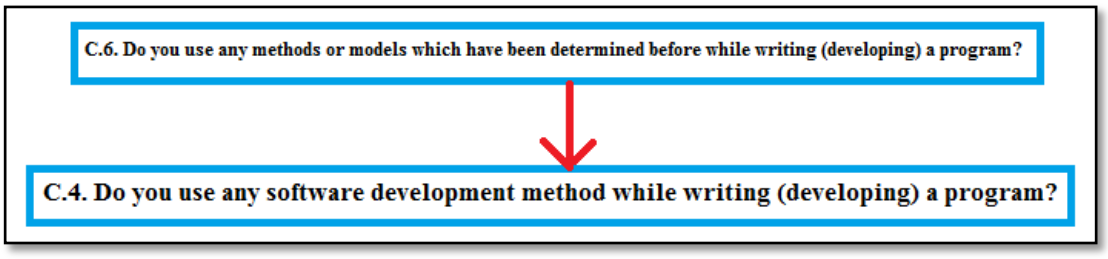

Figure 6.4 The relation of developed methods to software development method

Based on the survey, about 95% of the participants who use some methods or models which have been determined before while developing a program, at the same time, they use a software development method while writing a program.

It can be seen that applying methods which have been determined before in program implementation diverts software developers to use a software development method in software progress. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

Using some methods or models which have been designed and implemented before by software programmers while developing a program shows that there is the property of Reusability in program development. This property is the main factor of applying a software development method in software progress since a development method provides software developers analyzed, tried and practiced methods and models which make their works easy while writing a program. That means the property of Reusability decrease the cost of the project in terms of working force and completion time.

In addition, it can't be ignored that the term "method" in the "C.6 question" may have been supposed to be the term "software development method" in the "C.4 question" by the participants who have taken the survey. Therefore, they may have answered the "C.4 and C.6 questions" in the same direction.

6.16 The Relation of User Requirements to Developed Methods

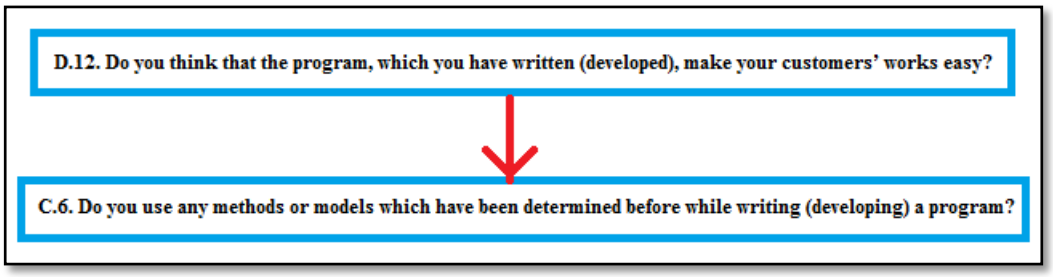

Figure 6.5 The relation of user requirements to developed methods

Based on the survey, about 97% of the participants who think that the program, which they have developed, make their customers' works easy, as well they use some methods or models which have been determined before while writing a program.

It can be shown that thinking that the developed program makes the customers' works easy by software programmers diverts them to use methods which have been determined before in program implementation. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

The software programmers who work methodically based on the software development process think that they master the user needs more than others, and design and develop a program which meets the user requirements and the project outputs almost completely.

6.17 The Relation of "Exception Handling" to "Error Handling"

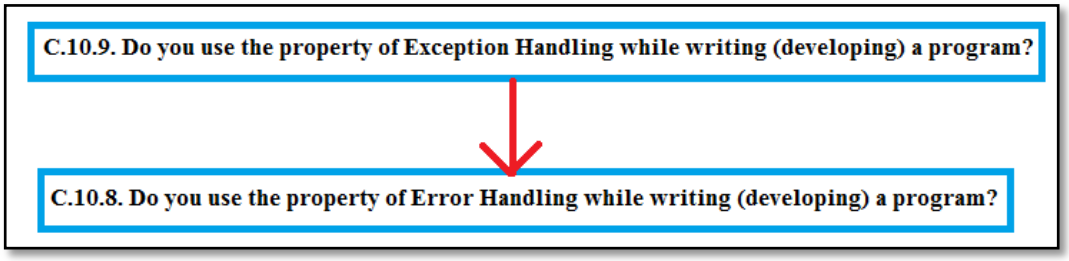

Figure 6.6 The relation of "exception handling" to "error handling"

Based on the survey, about 88% of the participants who use the property of "Exception Handling" while developing a program, at the same time, they use the property of "Error Handling" while writing a program.

It can be determined that applying the property of "Exception Handling" in program implementation diverts software programmers to use the property of "Error Handling" in program development. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

Based on the relation tree, the property of "Exception Handling" is the main condition of the property of "Error Handling". That means if a software developer would like to do error handling operation while programming, s/he has to do exception handling operation before that.

6.18 The Relation of Program Quality to Reusability

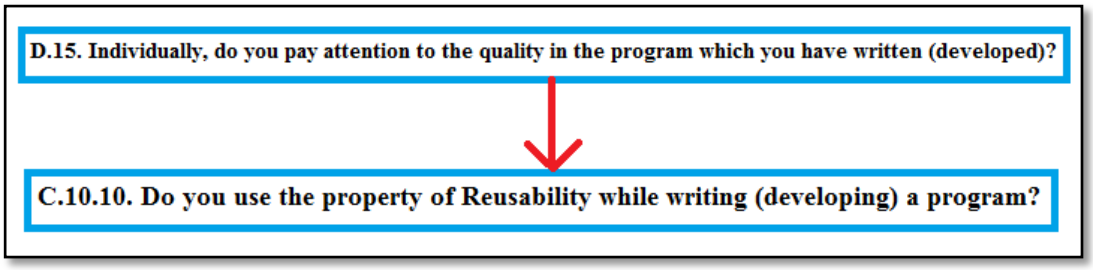

Figure 6.7 The relation of program quality to reusability

Based on the survey, about 95% of the participants who pay attention to the quality in the program which they have developed, as well they use the property of "Reusability" while writing a program.

It can be indicated that paying attention to the quality in the program by software developers in program implementation diverts them to use the property of "Reusability" in program development. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

The software developers who pay attention to the quality while designing and implementing a program, can also apply the property of "Reusability" features easily at the same time since software quality depends on a software development method and this development process bring to reuse designed and developed methods and models with it.

6.19 The Relation of Reliable Code Writing to Practical Program

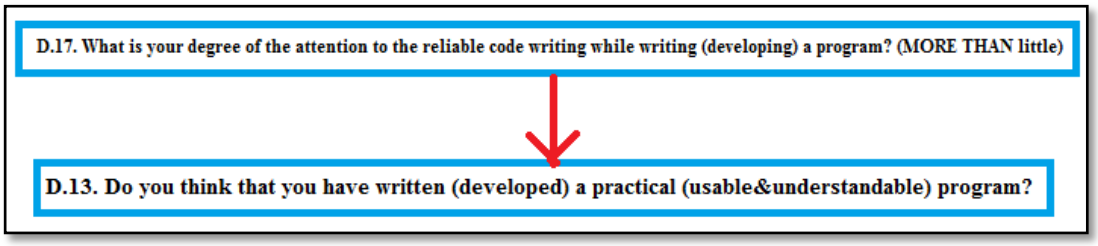

Figure 6.8 The reliable code writing to practical program

Based on the survey, about 92% of the participants who pay attention to the reliable code writing more than little while developing a program, at the same time, they think that they have written a usable and understandable program.

It can be seen that paying attention to the reliable code writing more than little by software developers in program implementation diverts them to think that they develop a practical program. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

Although there is not a connection between "reliable code writing" and "usable and understandable program" directly, some ideas can be generated about that:

- If a program can be executed, this may be understood and used.
- Since the participants who have taken the survey have enough technical knowledge, they are sure that an executed program should be understood and used. Therefore, they may have a wrong idea about that.

- If the participants who have taken the survey develop modules and sub-classes which are infrastructure oriented and used by software developers, then since their customers are software programmers, executable codes are accepted as understandable and usable.

6.20 The Relation of Developed Methods to Software Quality

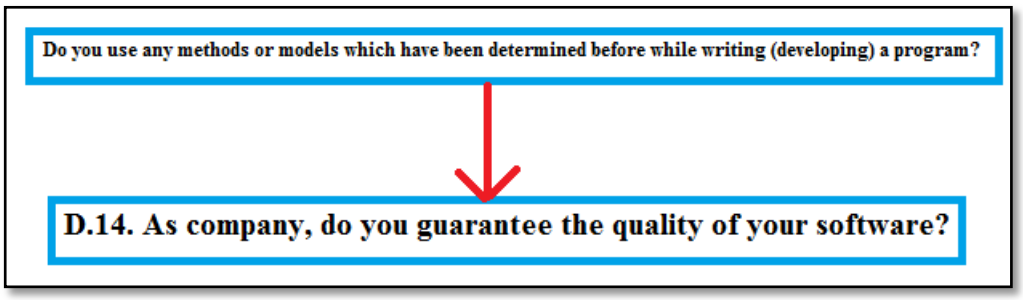

Figure 6.9 The relation of developed methods to software quality

Based on the survey, about 95% of the participants who use some methods or models which have been determined before while developing a program, as well their companies guarantee the quality of their software.

It can be shown that using models which have been determined before in program development diverts software programmers' companies to guarantee the quality of the software. The relationship among these criteria is in the same direction which means if the one is going up, and then the other one is going up, or if the one is going down, and then the other one is going down.

The models and methods which have been designed and developed before is accepted as a main factor of software quality since these models and methods are required from software development methods which have an aim of rising the quality in software products.

CHAPTER SEVEN
CONCLUSION AND FUTURE WORK

In this master thesis, the background about the work has been prepared before the study of thesis by observing and analyzing the researches which have been done before and the case studies which have been published before about the thesis subject. With this background study, the common criteria set about the measurement and evaluation of software developers' performance has been presented. Some information has been got from some software developers and managers by doing survey technic on internet so as to evaluate the use of the common criteria in real work life and identify criteria which are used in real work life but not seen in researches before. In the light of the survey results, a measurement and evaluation criteria set about the software developers' performance have been created by applying an association rule mining algorithm - Apriori - and a decision tree algorithm - C4.5. Furthermore, this master thesis provides reliable perspective to authorized people about how to measure and evaluate software developers' performance while they are working in a software project.

This master thesis shows that there are mainly six relations in the criteria set about the measurement and evaluation of software developers' performance: between developed methods and software development methods, between user requirements and developed methods, between the property of "exception handling" and the property of "error handling", between program quality and reusability, between reliable code writing and practical (usable and understandable) program and between developed methods and software quality.

Some attachments may be done to increase reliability of measurement and evaluation criteria set in this work. For instance, objective and automatic measurement and evaluation has to be supported by using some tools and applications in software developers' working computers since now, the declarations of the participants, who have taken the survey, are the base case of this thesis work

and they are accepted as true. In addition, the quantitative data have to be fixed so as to measure and evaluate qualitative data more reliably.

"You can't manage the process which you don't measure." This statement which is claimed to be said by Peter DRUCKER shows that the software development process has to be measured by clear and objective variables. Thus, some reliable data are specified and determined so that this software development process can be managed by benefiting from these trustworthy results. According to the results of this evaluation, manpower, what is the main resource of software development process will be used more effectively. And then, the benefits of the "Software Engineering" may be seen more tangible.

REFERENCES

Abstraction (computer science). (n.d.). Retrieved May 1, 2015, from http://en.wikipedia.org/wiki/Abstraction_(computer_science)

Algorithmic efficiency. (n.d.). Retrieved May 2, 2015, from http://en.wikipedia.org/wiki/Algorithmic_efficiency

Baggelaar, H. (2008). *Evaluating programmer performance visualizing the impact of programmers on project goals.* M.Sc Thesis, University of Amsterdam, Amsterdam.

Balijepally, V., Nerur, S., & Mahapatra, R. (2012). Effect of task mental models on software developer's performance: An experimental investigation. *45th Hawaii International Conference on System Science, IEEE,* 5442 - 5451.

Calikli, G., & Bener, A. (2013). An algorithmic approach to missing data problem in modeling human aspects in software development. *Proceedings of the 9th International Conference on Predictive Models in Software Engineering, ACM,* Article No. 10.

Calikli, G., & Bener, A. (2010). Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance. *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM,* Article No. 10.

Chilton, M. A., Hardgrave, B. C., & Armstrong, D. J. (2010). Performance and strain levels of it workers engaged in rapidly changing environments: A person-job fit perspective. *ACM SIGMIS,* 8 - 35.

Coupling (computer programming). (n.d.). Retrieved May 3, 2015, from http://en.wikipedia.org/wiki/Coupling_%28computer_programming%29

Duarte, C. B., Faria, J. P., & Raza, M. (2012). PSP PAIR: Automated personal software process performance analysis and improvement recommendation. *Eighth International Conference on the Quality of Information and Communcations Technology, IEEE,* 131 - 136.

Ehrlich, K., & Cataldo, M. (2012). All-for-one and one-for-all?: A multi-level analysis of communication patterns and individual performance in geographically distributed software development. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work,* 945 - 954.

Error handling definition. (n.d.). Retrieved May 4, 2015, from http://searchsoftwarequality.techtarget.com/definition/error-handling

Exception handling. (n.d.). Retrieved May 5, 2015, from http://en.wikipedia.org/wiki/Exception_handling

Gallivan, M. J. (1998). The influence of system developers' creative style on their attitudes toward and assimilation of a software process innovation. *Proceedings of the Thirty-First Hawaii International Conference on System Sciences, IEEE,* 435 - 444.

Golb, O. (2013). *C4.5 decision tree implementation.* Retrieved June 1, 2015, from http://www.otnira.com/2013/03/25/c4-5/

Hall, T., Wilson, D., Rainer, A., & Jagielska, D. (2007). The neglected technical skill? *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce,* 196 - 202.

Inheritance (object-oriented programming). (n.d.). Retrieved May 6, 2015, from http://en.wikipedia.org/wiki/Inheritance_%28object-oriented_programming%29

Kelly, B., & Haddad, H. M. (2012). Metric techniques for maintenance programmers in a maintenance ticket environment. *Journal of Computing Sciences in Colleges, ACM, 28* (2), 170 - 178.

Kharytonov, V. (2012). *Software measurement: Its estimation and metrics used.* Retrieved May 1, 2015, from http://it-cisq.org/software-meausrement-estimation-metrics/

Layout (computing). (n.d.). Retrieved May 7, 2015, from http://en.wikipedia.org/wiki/Layout_%28computing%29

Lee, K., Joshi, K., & Kim, Y. (2008). Person-job fit as a moderator of the relationship between emotional intelligence and job performance. *Proceedings of the 2008 ACM SIGMIS CPR Conference on Computer Personnel Doctoral Consortium and Research,* 70 - 75.

Michael, R. S. (2008). *Measurement*. Retrieved April 1, 2015, from http://www.indiana.edu/~educy520/sec5982/week_3/measurement_rsm.pdf

Mining frequent itemsets - apriori algorithm. (n.d.). Retrieved May 1, 2015, from http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab8-Apriori.pdf

Modularity. (n.d.). Retrieved May 8, 2015, from http://en.wikipedia.org/wiki/Modularity

Ramler, R., Klammer, C., & Natschläger, T. (2010). The usual suspects: A case study on delivered defects per developer. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement,* Article No. 48.

Reliability, availability and serviceability (computing). (n.d.). Retrieved May 9, 2015, from

http://en.wikipedia.org/wiki/Reliability,_availability_and_serviceability_(computing)

Reusability. (n.d.). Retrieved May 10, 2015, from http://en.wikipedia.org/wiki/Reusability

Robustness (computer science). (n.d.). Retrieved May 11, 2015, from http://en.wikipedia.org/wiki/Robustness_(computer_science)

Sawyer, S., & Guinan, P. J. (1998). Software development: Processes and performance. *IBM Systems Journal, 37* (4), 552 - 569.

Schröter, A., Aranda, J., Damian, D., & Kwan, I. (2012). To talk or not to talk: Factors that influence communication around changesets. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work,* 1317 - 1326.

Software maintenance. (n.d.). Retrieved May 12, 2015, from http://en.wikipedia.org/wiki/Software_maintenance

Software portability. (n.d.). Retrieved May 13, 2015, from http://en.wikipedia.org/wiki/Software_portability

Software testability. (n.d.). Retrieved May 14, 2015, from http://en.wikipedia.org/wiki/Software_testability

Test coverage in technology. (n.d.). Retrieved May 15, 2015, from http://dictionary.reference.com/browse/test+coverage

The software development process - evaluation. (n.d.). Retrieved May 16, 2015, from http://www.slideshare.net/cunniman/6-the-software-development-process-evaluation-2871704

Thing, C. (2008). The application of the function point analysis in software developers' performance evaluation. *4th International Conference on Wireless Communications, Networking and Mobile Computing, IEEE,* 1 - 4.

Wang, Y., & Zhang, M. (2010). Penalty policies in professional software development practice: A multi-method field study. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering,* 39 - 47.

Westermann, D. (2012). A generic methodology to derive domain-specific performance feedback for developers. *34th International Conference on Software Engineering, IEEE,* 1527 - 1530.

What is evaluation? (n.d.). Retrieved Apri1 2, 2015, from ftp://ftp.hrsa.gov/hab/m4p1.pdf

What is layering? (n.d.). Retrieved May 18, 2015, from http://whatis.techtarget.com/definition/layering

What is software measurement and analysis? (n.d.). Retrieved May 19, 2015, from http://www.sei.cmu.edu/measurement/index.cfm

Zhang, S., Wang, Y., & Xiao, J. (2008). Mining individual performance indicators in collaborative development using software repositories. *15th Asia-Pacific Software Engineering Conference, IEEE,* 247 - 254.

APPENDICES

APPENDIX 1: Companies

The companies where the participants, who have taken the survey about this master thesis, work:

- 2AG Yazılım Hizmetleri Ltd. Şti.
- Accenture
- Alcatel-Lucent
- Anadolu Sigorta
- AR Pandora
- Arçelik
- Bilgi Güvenliği Akademisi
- Buzz Dijital
- Control4
- Cronom
- Desmer
- DOGO
- Ege Üniversitesi Tıp Fakültesi Hastanesi
- Epigra
- Ericsson
- Etiya
- Finansbank
- ForceData
- Garanti Teknoloji
- Gediz Üniversitesi Bilgi İşlem Müdürlüğü
- Huawei
- INENSUS
- ING Bank
- İnnova Bilişim Çözümleri A.Ş.
- izmo Bilişim

- Kentkart
- Kontrol Bilgi Sistemleri
- Lynx S.p.A.
- Mavikent Bilişim
- Metadata Bilişim
- MilSOFT
- OBSS
- Orman Genel Müdürlüğü
- philogica
- Pikotek Arge İnovasyon Enerji San. Tic. A.Ş.
- Research Group CAMMA
- Salihli Belediyesi Bilgi İşlem Müdürlüğü
- Semafor Teknoloji
- Siemens
- Siskon Yazılım ve Otomasyon
- SoftTech
- Trendyol
- Turkcell
- TÜRKİYE HALK BANKASI A.Ş.
- Türkiye İlaç ve Tıbbi Cihaz Kurumu / Sağlık Bakanlığı
- University College Cork (UCC)
- Ünipa A.Ş.
- Ünivera Bilgisayar Sistemleri A.Ş.
- Vakıfbank Elektronik Bilgi İşlem Sistemleri
- VERİPARK
- Vestel
- Vodafone
- WebAçık Hosting
- Yapı Kredi

APPENDIX 2: The Survey

Yazılım Geliştiricilerinin Çalışma Performanslarının Ölçülmesi ve Değerlendirilmesine Yönelik Kıstaslar

"Hayatta En Hakiki Mürşit İlimdir, Fendir." Gazi M. Kemal ATATÜRK / 22 Eylül 1922

Yazılım sektörü; kişiler, kültürler ve firma uygulamaları başta olmak üzere birçok parametreye bağlı olarak değerlendirilmesi son derece karmaşık ve bir o kadar da önemli bir sektör olarak görünmektedir. Aşağıda bulunan anket çalışması, sizin gibi Türkiye ve Dünya çapında Yazılım geliştirmenin farklı aşamalarında görev yapmakta olan birçok ilgili Mühendis ile paylaşılmıştır.
Amacımız, Yazılım sektöründe çalışan sizlerin Performans değerlendirilmesi sırasında kullandığınız bir takım verilerinizin arasındaki ilişkileri ortaya koymak suretiyle değerlendirme süreçlerinin daha objektif tanımlanabilmesi için bir yöntem geliştirebilmektir.

Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği bölümünde yürütülmekte olan, Araş. Gör. Mustafa BATAR'a ait Yüksek Lisans tez verilerini oluşturacak çalışmamıza vermiş olduğunuz destekten dolayı şimdiden çok teşekkür ederiz. (Yard. Doç. Dr. Kökten Ulaş BİRANT)

A. Genel Bilgiler

A.1. Yaşınız kaçtır?

- ☐ 15-20
- ☐ 20-25
- ☐ 25-30
- ☐ 30-35
- ☐ 35-40
- ☐ 40-...

A.2. Eğitim seviyeniz nedir? (En son mezun olunan)

- ☐ Lise ve altı
- ☐ Lisans
- ☐ Yüksek lisans
- ☐ Doktora

A.3. Eğitim türünüz nedir?

- ☐ Bilgisayar / Yazılım / Bilgisayar bilimleri / ... mühendislikleri
- ☐ Diğer mühendislik alanları
- ☐ Matematik / İstatistik
- ☐ Programlama

- ☐ Diğer

B. Program Yazarken (Geliştirirken) ...

B.1. Yorum satırları hariç günde ortalama kaç satır kod yazıyorsunuz?
Satır sayısını yazınız.

B.2. Günde ortalama kaç defa yazdığınız kodu uygulamaya sokuyorsunuz? (Derleme İşlemi)
Uygulama (derleme işlemi) sayısını yazınız.

B.3. Günde ortalama kaç defa önceden yazmış olduğunuz kodda değişiklik yapıyorsunuz?
Değişiklik sayısını yazınız.

B.4. Günde ortalama kaç satır yorum yazıyorsunuz?
Satır sayısını yazınız.

B.5. Program yazarken (geliştirirken) günde ortalama kaç tane sınıf yazıyorsunuz?
Sınıf sayısını yazınız.

B.6. Yazdığınız (geliştirdiğiniz) bir programda günde ortalama kaç tane metot yazıyorsunuz?
Metot sayısını yazınız.

B.7. Program yazarken (geliştirirken) günde ortalama kaç tane modül/sınıf bitirebiliyorsunuz?
Modül/Sınıf sayısını yazınız.

B.8. Yazdığınız (geliştirdiğiniz) bir programda günde ortalama kaç tane hata tespit edebiliyorsunuz?
Tespit edilen hata sayısını yazınız.

B.9. Yazdığınız (geliştirdiğiniz) bir programda günde ortalama ne kadar hatayı düzeltebiliyorsunuz?
Düzeltilen hata sayısını yazınız.

B.10. Yazdığınız (geliştirdiğiniz) bir programda eğer bir hata var ise hatayı telafi etme süreniz ortalama ne kadardır?
Saat sayısını yazınız.

B.11. Yazdığınız (geliştirdiğiniz) bir programda günde ortalama kaç tane düzelttiğiniz hata tekrardan hata veriyor?
Tekrar eden hata sayısını yazınız.

B.12. Program yazarken (geliştirirken) günde ortalama kaç tane birim test yazıyorsunuz?
Birim test sayısını yazınız.

C. Tasarım ve Yönetim

C.1. Program yazarken (geliştirirken) sabit kod parçaları kullanıyor musunuz?

- ☐ Evet
- ☐ Hayır

C.2. Her geliştirilen (geliştirdiğiniz) fonksiyon/modül kullanıcıya doğrudan iletilmekte midir yoksa birleştirilip belirli periyotlarla paket halinde mi sunulmaktadır?

- ☐ Fonksiyon/Modül tek tek
- ☐ Fonksiyon/Modül paket halinde

C.3. Program yazarken (geliştirirken) değişken, metot ve sınıf isimleri konusunda belirlenmiş kurallarınız var mıdır?

- ☐ Evet
- ☐ Hayır

C.4. Program yazarken (geliştirirken) herhangi bir yazılım geliştirme yöntemi kullanıyor musunuz?

- ☐ Evet
- ☐ Hayır

C.5. Otomatikleştirilmiş bir yazılım geliştirme yönteminiz var mıdır?

- ☐ Evet
- ☐ Hayır

C.6. Program yazarken (geliştirirken) önceden belirlenmiş yöntem ya da örnekleri kullanıyor musunuz?

- ☐ Evet
- ☐ Hayır

C.7. Sizin için paylaşılan bir kod deposu ya da kütüphanesi var mıdır?

- ☐ Evet
- ☐ Hayır

C.8. Program yazarken (geliştirirken) önceden yazdığınız kodları sık sık değiştirme ihtiyacı duyuyor musunuz?

- ☐ Evet
- ☐ Hayır

C.9. "Inheritance" özelliğini kullanıyorsanız bunun derinlik derecesi kaçtır?
Derece miktarını yazınız.

C.10. Program yazarken (geliştirirken) aşağıdaki özellikleri kullanıyor musunuz?

	Evet	Hayır

	Evet	Hayır
"Abstraction"	○	○
"Testability"	○	○
"Use of Layers"	○	○
"Layouts"	○	○
"Modularity"	○	○
"Coupling"	○	○
"Test Coverage"	○	○
"Error Handling"	○	○
"Exception Handling"	○	○
"Reusability"	○	○

D. Yazdığınız (Geliştirdiğiniz) Programı Tamamlama Süreci

D.1. Program yazarken (geliştirirken) günde ortalama kaç saat çalışıyorsunuz?

Saat sayısını yazınız.

D.2. Bir programı yazmaya (geliştirmeye) başlamadan önce bunu düşünme süreniz ortalama ne kadardır?

Saat sayısını yazınız.

D.3. Bir programı bitirme süreniz ortalama ne kadardır?

Saat sayısını yazınız.

D.4. Genel olarak, yazdığınız (geliştirdiğiniz) bir programı tekrar gözden geçirme süreniz ortalama ne kadardır?

Saat sayısını yazınız.

D.5. Genel olarak, yazdığınız (geliştirdiğiniz) programlarda erken teslim yapıyorsanız bu süre ortalama ne kadardır?

Saat sayısını yazınız.

D.6. Genel olarak, yazdığınız (geliştirdiğiniz) programlarda ortalama ne kadar süreli bir gecikme yapıyorsunuz?

Saat sayısını yazınız.

D.7. Genel olarak, program yazarken (geliştirirken) ortalama ne kadar süreli bir zamanlama hatası yapıyorsunuz?

Saat sayısını yazınız.

D.8. Genel olarak, program yazarken (geliştirirken) ortalama ne kadar büyüklükte bir boyutlama hatası yapıyorsunuz?

MB miktarını yazınız.

D.9. Genel olarak, program yazarken (geliştirirken) analizde belirtilmemiş ortalama kaç tane özellik/fikir kodluyorsunuz? (Örneğin; bakım, raporlama vb. ile ilgili)

Kodlanan özellik/fikir sayısını yazınız.

D.10. Genel olarak, program yazarken (geliştirirken) ortalama ne kadar büyüklükte bir bölümleme hatası yapıyorsunuz? (Örneğin; geliştirilen bir özellik için: "tasarımda belirtilen fonksiyon sayısı" - "geliştirilen fonksiyon sayısı")

Fark miktarını yazınız.

D.11. Program yazarken (geliştirirken) analizde belirtilen çıktılara ne ölçüde cevap verebiliyorsunuz?

- ☐ Çok az
- ☐ Az
- ☐ Orta
- ☐ Fazla
- ☐ Çok Fazla

D.12. Yazdığınız (geliştirdiğiniz) bir programın müşterilerinizin işlerini kolaylaştırdığını düşünüyor musunuz ve buna inanıyor musunuz?

- ☐ Evet
- ☐ Hayır

D.13. Kullanması kolay/anlaşılabilir bir program yazdığınızı (geliştirdiğinizi) düşünüyor musunuz?

- ☐ Evet
- ☐ Hayır

D.14. Firma olarak yazılım kalitesi ile alakalı güvence/teminat veriyor musunuz?

- ☐ Evet
- ☐ Hayır

D.15. Kişisel olarak yazdığınız (geliştirdiğiniz) programda kaliteye dikkat ediyor musunuz?

- ☐ Evet
- ☐ Hayır

D.16. Yazdığınız (geliştirdiğiniz) programın kalitesi nasıldır?

- ☐ Çok kötü
- ☐ Kötü
- ☐ Orta
- ☐ İyi
- ☐ Çok iyi
- ☐ Değerlendirmiyorum

D.17. Program yazarken (geliştirirken) güvenli kod yazımına ne kadar dikkat ediyorsunuz?

- ☐ Çok az
- ☐ Az
- ☐ Orta
- ☐ Fazla
- ☐ Çok fazla

D.18. Yazdığınız (geliştirdiğiniz) bir program tamamlandıktan sonra müşterilerinize teknik desteği siz mi (bireysel olarak) veriyorsunuz?

- ☐ Evet
- ☐ Hayır

D.19. Geliştirme süreciniz, projenin kullanıcıya teslim sonrası yeniden gözden geçirilmesini içeriyor mu?

- ☐ Evet
- ☐ Hayır

E. Program Yazarken (Geliştirirken) İçinde Bulunulan Ortam

E.1. Yazdığınız (geliştirdiğiniz) programların eğitim/beceri düzeyinize göre ortalama zorluk derecesini nasıl değerlendirirsiniz?

- ☐ Çok kolay
- ☐ Kolay
- ☐ Orta
- ☐ Zor
- ☐ Çok zor

E.2. Programlama yaparken yalnız mı çalışıyorsunuz yoksa takım halinde mi?

- ☐ Yalnız
- ☐ Takım halinde

E.3. Çift halinde mi programlama yapıyorsunuz?

- ☐ Evet
- ☐ Hayır

E.4. Programlama yaparken yaratıcılığınızı kullanma fırsatınız oluyor mu?

- ☐ Evet
- ☐ Hayır

E.5. Program yazarken (geliştirirken) özgün müsünüz?

- ☐ Evet
- ☐ Hayır

E.6. Kullanılan yazılım geliştirme aracınızın değişmesinin programa etkisi nasıl olurdu?

- ☐ Çok az

- Az
- Orta
- Fazla
- Çok fazla

E.7. Yazdığınız (geliştirdiğiniz) bir program tamamlandıktan sonra eğer bir değişiklik yapılacaksa program bundan ne ölçüde etkileniyor?

- Çok az
- Az
- Orta
- Fazla
- Çok fazla

E.8. Yazılım geliştirme sürecinde işleyişe bağlı mısınız?

- Evet
- Hayır

E.9. Sizden yazdığınız (geliştirdiğiniz) programla ilgili bir rapor isteniyor mu?

- Evet
- Hayır

E.10. Çalışmakta olduğunuz firmanızda belirli bir raporlama sistemi var mıdır?

- Evet
- Hayır

E.11. Değerlendirme sonuçlarını sizinle paylaşmamızı istiyorsanız e-posta adresinizi girebilirsiniz.

E.12. Kaç farklı firmadan veri alındığını yorumlamamız açısından şu anda çalışmakta olduğunuz firmanızın ismini girebilir misiniz? (NOT: Bu alanın doldurulması ZORUNLU DEĞİLDİR.)