DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPING A TOOL FOR ANALYSIS OF SOFTWARE AND EVALUATION OF SOFTWARE DEVELOPER

by

Elnur SAFARLI

June, 2013

İZMİR

DEVELOPING A TOOL FOR ANALYSIS OF SOFTWARE AND EVALUATION OF SOFTWARE DEVELOPER

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

by

Elnur SAFARLI

June, 2013

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DEVELOPING A TOOL FOR ANALYSIS OF SOFTWARE AND EVALUATION OF SOFTWARE DEVELOPER"** completed by **ELNUR SAFARLI** under supervision of **ASSIST. 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.

.....

Asist. Prof. Dr. Kökten Ulaş BİRANT

Supervisor

.....

(Jury Member)

.....

(Jury Member)

Prof. Dr. Ayşe OKUR

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to acknowledge to my supervisor Asist. Prof. Dr. Kökten Ulaş BİRANT for his assistance and guidance in the preparation of this thesis. He spent most of his times helping me and analyzing of my thesis results.

I give to thank Dokuz Eylul University for providing me with web server and other facilities. I also thank to Global Programmers Club's members for giving me program's source code.

DEVELOPING A TOOL FOR ANALYSIS OF SOFTWARE AND EVALUATION OF SOFTWARE DEVELOPER

ABSTRACT

The objective of this study was to develop measurement of software quality factors. Most of Software measures were defined many years ago but now in industry they are not used widely. These type of measures were used for determination of user satisfaction. And it means that the idea of user satisfaction is not new. Satisfying of user is more important in modern business and developing technology. Software metrics play an important role in determination of quality factors.

Definition and calculation of Halstead's method and LOC metrics are analyzed in this thesis. These analysis are based on measurement analysis that define program quality factors. Difference of these analysis are measuring the program's metrics without seeing source code. After measurement the quality factors are determined by web tool.

Keywords : Software quality factors, software metrics, Halstead's metric, Line of Code (LOC) metrics, software measurement, quality in use.

YAZILIM ANALİZİ VE YAZILIM GELİŞTİRİCİNİN DEĞERLENDİRİLMESİ AMACIYLA BİR ARAÇ GELİŞTİRİLMESİ

ÖZ

Bu çalışmanın amacı, yazılım kalite faktörleri ölçümünü geliştirmektir. İlgili yazılım parametrelerinin çoğu yıllar önce tanımlanmış ancak şu anda endüstride yaygın olarak kullanılmamaktadır. Bu ölçümler doğrudan veya dolaylı olarak kullanıcının memnuniyetini oluşturmak için kullanılır. Kullanıcı memnuniyeti uzun zamandır önemli bir kriter olmasına rağmen, memnuniyeti geliştirmeye yönelik bu objektif değerlendirme sistemlerinde önemli bir gelişme yaşanmamıştır. Kullanıcının memnuniyeti modern girişimcilikte ve teknoloji gelişmesinde önemli yere sahiptir ve yazılım ölçütleri kalite faktörlerinin belirlenmesinde önemli rol oynamalıdır.

Bu tezde Halstead ve LOC ölçümleri ve hesaplamaları analiz edilmektedir. Bu analiz programı kalite faktörleri tanımlayan ölçüm analizine dayanır. Bu analizin diğer analizlerden farkı kaynak kodu görmeden program ölçümleri ölçmesidir. Ölçüm yapıldıktan sonra kalite faktörleri web aracı tarafından belirlenir ve raporlanarak tez dahilinde yorumlanır.

Anahtar sözcükler : Yazılım kalite faktörleri, yazılım ölçütleri, kod satrı ölçümleri halstead ölçümleri, yazılım ölçümü, kalite kullanımı.

CONTENTS

Page

LIST OF FIGURES

Pages

LIST OF TABLES

Pages

CHAPTER ONE

INTRODUCTION

1.1 Scope

In modern time users demand powerful and reliable software. Companies want to control software complexity and development process. Most computer scientists and experts agree that the complexity of program will increase in future. But it takes more time to define and measure complex program.

Software metrics are defined from program numbers through to model that help for determination of software quality. It is known that all methods measure the metrics of program by using this number. Program measures have an application to estimating of software products. They are used to calculate productivity and complexity of products. This thesis presents method of Halstead's and LOC which determine quality factors of software. A number of measures in industry are not well understood. Halstead and LOC measures were proposed nearly thirty years ago. To definition of Halstead's metric, firstly defining the context and designing the measurement must be applied.

The quality factors are quantified by SQM. The developers and project managers determine these factors because it is more important for their applications. Quality is measured by software metrics. Many program metrics correlate with these factors. Our thesis's the web tool uses a metric that based on source code that is used in avionic product. Because this type of metrics is not changeable, it can be used and measure in any time. In this research only product qualities factor are measured. These types of quality are based on source code and program metrics. So Halstead and LOC metrics are applied to source code for determination. In measurement the web tool can not see the source code of program.

1.2 Audience

The objective of our thesis is to provide creating new method that helps organizations and project managers to determine quality of program without seeing source code. Developers, project managers, testers and product improvement teams should get our tools helpful in measurement of software metrics and collecting data. There are many reports or analysis tools that measure the quality by seeing source code.

The determination of software quality factors based on 318 source code from 36 countries. The working principle of tool is guided by Halstead metrics and LOC metrics. In modern programming the using of this types metric is not so widely. But in thesis it is seen that the quality factor can be determined by using the these metrics.

1.3 Structure of Thesis

Chapter 2 discusses the quality and analysis of software. This chapter defines the SQM, discusses about the measurement methods . In this chapter the type of metrics and quality factors are explained.

Chapter 3 defines measurement of Halstead’s metrics and its calculation. The equation of Halstead's metric and calculation are presented. Near Halstead metrics LOC is also define in this chapter. The advantages and calculation of this metric are also represented in this section. At last of this chapter the relation between quality factors and these type of metrics are described.

Chapter 4 describes the working principles of web tools and the collecting of data from programmers. The analyzing of program and developers quality is also presented in this chapter. The qualities are defined by programming languages and countries. These comparisons are shown in pie charts.

After these chapters the conclusion discuss the goal of web tool and which innovation we do in this thesis.

CHAPTER TWO

QUALITY AND ANALYSIS OF SOFTWARE

Definition and means of Software quality, about its importance, software metrics type and their meanings and functions take main part of this chapter. There are some quality metrics that are used for analysis of software. The qualities of software or products are determined by these quality factors.

2.1 Definition of SQ and Its Importance

Computer systems generally based on two things. They are hardware and software. The hardware consists of physical parts of computers and the software consists of programs. The code is a subset of software. It is used for the purpose of being loaded into computer to control it. If it can be read by people then it is called source code. The translation of source code is loaded into the computer is called object code.

The first definition of quality in software science is said by Shewhart at the beginning of 20th century: "There are two common aspects of quality: one of them has to do with the consideration of the quality of a thing as an objective reality independent of the existence of man. The other has to do with what we think, feel or sense as a result of the objective reality. In other words, there is a subjective side of quality."(Wikipedia) But now there are many definitions of software quality that defined by scientists. And, all of these are not also been satisfactory by software industry.

1. Software quality is determined by a set of quality factors.

2. Software quality is determined by the user satisfaction.

3. Software quality is determined by the errors or unexpected behavior of the software.

All of these items of these are significance for software quality, since they are important for determination of software quality. The software quality are divided into two parts: Software functional quality and Software structural quality.

- Functional quality generally based on complies and design of software, functional requirements and etc.

- Structural quality is defined through the analysis of software's inner structure. Source code, architecture of software and etc. effect for determination of this quality type.

Difference between structural and functional quality is the functional quality that is evaluated through software testing. Every time it has been more difficult to define software quality. It is reason that the comprehend of quality is change person to person and object to object. The description of software quality is differ among project manager, tester, user clients, developers. academics and quality consultants.

The SQM stands for a quality modifies software. SQM measures the quality of software. This measure is based on quality factors which is defined when it affects shown objectives such as software reliability and complexity.

Another measure's type is metric. The measurement is based on relations of numbers. These measures are monotonic and repeatable. In repeatable measurement the results never change and anyone can get same results at any time. In the monotonic measurement, the certain change in the measure always represents a certain change in the property being measured. Although such a function is consistently non increasing or non decreasing, it is not proportional. It is a desirable measure and is reflected in the measure by the mathematical relation. This measure is considered well-behaved.

2.2 Software Metrics

Software metrics are necessary in computer science because there are many companies that used software product on their work. Every director of company wants that the software which they use will be reliable. The measurement of quality

is determined how software product is reliable. For that we use metrics to know quality factors and measure characteristics of software. The metrics must apply to software's code and avionics equipment code. Notwithstanding factors are not correlated with metric in some time, it helps programmers to develop their programs. Every project manager knows that without metrics there is no any way to determine improving of software quality. In software engineering the "software metrics" are related with the measurement. Therefore without measure we cannot determine the quality of software. Before measuring the source code of software it must be translated into numbers. There are several areas that are used in software metrics. These areas are used in the planning of improvement of software. Software metrics are used for developing of software size, in complexity analyzing, in defect analyzing and in effort estimation.

2.2.1 Classification of Software Metrics

There are different measures during the coding of software. So the software metrics grouped as follow.

a) Process Metrics

b) Product Metrics

c) Resource Metrics

2.2.1.1 Process Metrics

General purpose of this thesis is determination the quality factors by using this metrics. Process metric is making a big role to understand programming process. Process conception differs from company or tester. Such companies define this metric as end product metrics. These metrics following the defect during machine testing. On the other hand some organizations use this type of software metrics on each phase of development cycle. There are several parts include this type of metric.

Defect density during machine testing is correlated with the defect rate on the field. If there are many defects during machine testing it will be harder to develop

software product. The higher defects of code are determined during software testing. After determination of defect developer must fix it until the product is given to user.

The metrics of defects per line of code or function point is determination of quality while software is tested. The project manager and tester use this metric for quality determination. If the defect rate during the testing is lower or same than the previous, then you must do more testing. But if the defects rate higher than previous one, then quality perspective is negative.

The phase defect density is more useful between metrics. This metric uses for all phase of development cycle. The design review, code analyzing is included into the this cycle. The most percentage of defects relates to the design problem. Because design never harmonize with the results of programming product. More organizations use this defect density metrics in design and coding phases for determination of effort in process quality.

After the testing we know the number of defects. Which must be removed the defect from the product . In this, we can calculate the defect removal efficiency.

$$\text{DRE} = \frac{DRDP}{DLP} * 100$$

- DRE: Defect Removal Efficiency

- DRDP: Defects removed during a development phase

- DLP : Defects latent in the product

The total number of DLP in any development phase is not known, it is usually estimated by DRDP + Defects Found Later. This metric calculate development process, for front end and for each phases. For that this metric is called early defect metric and phase efficiency.

The effort and cost of products are measured by this metrics type. Effort and cost are main quality metrics in the quality factors. Because cost metrics effect the price of product. We also determine some quality factors from effort metrics.

Software process metrics help developers to resolve future products problems. Software Process metrics also measure the cost and production time of software products. This metrics is known as management metrics and project manager can determine the future of program. So process metrics are making a big role in level of measurement.

2.2.1.2 Product Metrics

Unlike the process metrics, the product metrics are used to measure properties of software programs. With product metrics we can determine quality factors such as reliability, usability, cost and size of products. Product metrics divided into two parts; intrinsic product quality and customer satisfaction.

Mean time to failure (MTTF) is one measurement unit of product metrics. It is most used in security critical systems. This type of metrics measure the time between two failures. What is the difference between the failure and defects? Their definitions and differences of this metric change per organizations. The following differences were given from IEEE American National Standards Institute (ANSI).

- An error is a human mistake that occur in software
- A fault is caused when the system run the function.
- A defect is a anomaly of product.
- A failure is a problem that occurs when product is ready and used by clients.

In this definition, there is not any differences between fault and defect. Some organizations and companies take these terms as synonymously. These terms also are taken as synonymously in the calculation of program's quality.

Measure the MTTF metric is more expensive. Because solving the failure of products firstly need to collect all problems and failure from testers and costumers. MTTF measure the time between the two failures that occur in software. Some tester companies think that this metrics only take money and there are no any advantages for products. Unless it is more expensive, this metric influence increasing the maintainability and reliability quality factors.

The second measurement of product metrics is defect density metric. Differences between DDM of process metrics and DDM of product metrics is only number of defect density is used in process metrics. In product metrics number of defect are divided to line of code and function point. Defect density metrics use numerator and denominator for measurement. Numerator is a size of software that shows the number defect for a time. It shows how many defects have in definite time. The denominator is number the thousand line of code (KLOC) or number of function point.

Line of code is a simple metric among the metrics. How this metrics are measured? The line of code gives us the size of program. This metric used for productivity of program. There are two types of counting method in this metric. First, some organizations counting only code's line in the program and show this in LOC. In second type counting method the comments, blanks and code's line are showed in LOC.

Other density metrics is calculating by number of function point (FP). What is function point? This metric was developed by Albrecht for determination the number of function in program. Function points determine the effort of program before development process. Calculation of this metric is harder. This metric was used in industry for estimation of function in software. It is also called as complexity software metric.

There are five function types in FP. Each type function's number is counted in program. Then all types' metrics is collected and assigned a weight. These types are following. "

- Number of external inputs
- Number of external outputs
- Number of logical internal files
- Number of external interface files
- Number of external inquiries " (U.S. Department of Transportation Federal Aviation Administration, 1991)

External inputs file is a data that is included by user and changes the internal files of program. Example of external input files are scanner form and purchase card. Examples of this function in modern programming are delete, add, insert and etc. This function classified as three levels: simple, average, or complex. Simple input function contains basic element types. The many data types included in the complex input function. If function is not simple and complex then it means that it is average input function. If input's data is file, then it is called an internal file. There are many problems in measurement of input functions. Each people count the data as different. So, firstly tester must attribute how and which type of data they will count.

External output functions control the data which goes to user or another application. Example to external output functions is printer reports and monitor. External output functions are also divided by 3 levels for its complexity: sample, average and complex. Sample output has a maximum two column format. There are multiple columns format in average output function. Complex output function has a tangled file format transformation. In this metrics all data types are used as output functions.

Logical internal file is group of data, hardware readable file or a file that is generated by applications or users. Example of this type of file is card files, memory reader files and etc. Database files also includes in this file. Sample, average and complex are complexity levels of logical internal files. Sample level contains one record and data types. There are many records and data types in complex level. The data which is not a sample and complex is in average level.

External Interface file is a same as complexity level of logical internal file. This file is entered by user a passed through application. If the module related with other module in module level then it is called a external interface file. On system level module show as the as input in application then it is called external input file.

Product metrics improve the quality of products, compare existing systems with new system. These metrics describes all properties of development process. So with this metric we can determine quality improvement of program.

2.2.1.3 Resource Metrics

Resource Metrics measure the characteristic of resources and systems which programs run on. Examples for this metrics are developer’s skills, system performance and etc. Some project manager and tester call these metric as project metrics.

2.2.2 Measurement of Software Metrics

Measurement defines as different. Let's explaining definition of the measurement. Before measurement properties of program must be attributed. Then properties of program are converted to number. After changing this number is categorized by quality metrics. And process from beginning to the end is called measurement of program.

After measurement, the quality metrics of program can be determined. The measurement divided into two parts: Direct and Indirect Measurement.

2.2.2.1 Direct Measurement

Direct measure is a number that we can take from source code without another function. Examples of direct measurement are LOC (line of code- measure number of size), duration (time of process), Defects (number of defects during testing) and effort (measure works of person by month)

Defects and LOC are explained in product and process metrics. What is time duration means? This metrics are known as two types. For some organizations measure a time from starting project until finishing. For another organizations or tester time is a duration when program start running till giving results. But in modern programming the running of program depends on user. The running duration of source code is called time metrics.

Effort is most usable metrics. This metrics determine both developer and software quality. By this metrics project manager determine the quality of developer. In next chapters we will explain about effort and its calculation by Halstead's method.

2.2.2.2 Indirect Measurement

Indirect measure is number that collets many indirect metrics for categorizing quality factors. By this method we can measure some statistics metrics. For example productivity of programmer per month or per size of code, defect density per time or size of code and etc. The comparison of line of code to month effort is a calculation of programmers productivity.

Many Quality factors are determined by this measurements method. In next chapter these quality factors will be discussed.

2.3 Software Quality Factors

Quality Factor determination is more important for application in process of testing**.** Many years ago many tester determined the programs quality as negative or positive. The developing of programming also affects the determination of software quality. Increasing number of programming languages and their functions cause the distribution of quality per category. Then all these categories are named as quality factors. Each quality factors has own name and calculation method. But many factors are nearly related or nonrelated to each other.

There are many metrics type that relate with the quality factors. But some of these metrics do not correspond with factors. Only indirect measurement is correlated with quality factors.

The following factors are more important factors in software engineering.

2.3.1 Accuracy

The accuracy of software products are depended on program's functions. If the result is wrong then accuracy of product will be low. For example. The type of input you show in program as integer. But user enters the value as real. So program give the results error. If you show inputs type as real then the accuracy of program will be so high. Sometimes programmer makes accuracy so high but in this moment program's size expanded. Accuracy is same category with efficiency and simplicity

2.3.2 Clarity

Clarity metrics show that how programmer understand a program. Clarity gives the reader about module, inputs and outputs. Clear program is easy for testing. So tester can understand the program easily. For getting this quality factor programmer must comment before some functions, write lines and make description of variables. By measuring clarity tester or developer can find problem's point easily. Clarity is opposite to complexity.

2.3.3 Completeness

Completeness shows the how program is fully finished and customer satisfied. Completeness covers broad area. This quality relates with other quality factors. Complete program determines all working parts of program. Incomplete program contains many missing variables, incorrect remarks, the function that give incorrect results and etc. Complete program firstly must have a accuracy. Most programmer write in the code incorrect data specifications and missing some variables. Tester inputs some data during testing. If program is completed tester will not need to generate extra input data. Completeness is depending on programming languages and developer.

2.3.4 Complexity

Complexity is most used term in SQM. Complexity sometimes used as quality factor and sometimes as decision making. If it used as decision making then it measure for using software metrics.

Complexity is composed as many part. Many time testers, scientist want to know how many process and products of software relates with complexity. Understandability or errors rated of software are determined by this quality metrics. Project manager can determine of program such as cost, time and effort for budget the program properly.

Complexity quality factors divided into 2 types : structural and psychological. Structural complexity is depended on software products. Psychological complexity determines how person understands the program.

Definition of complexity relates by the time. Static metrics measure the product at given time, history metrics measure the product and process in time interval. Static metrics has 3 types of measurement ways. Volume, control organizations and data organizations. Volume measures the size of product. Control organizations measure structure of control. Data organizations measure the relation between program and data.

Complexity in many time related with maintainability. It is harder maintain when programs complexity is higher. Because complex program tends to more bug and it takes more time to fix these bugs. Complexity also conflicts with simplicity, understandability and etc.

2.3.5 Conciseness

Conciseness is ability of program that is using main function. Comments show the understandability of program but many comments confuse the attention of reader. Generally complexity is defined as structural complexity. Hence conciseness is inversely proportional to structural complexity. If program conciseness so it will be so cryptic and person ability cannot understand it fluently. This means that conciseness is proportional to complexity. We can understand from this proportion that when program is conciseness, it may not be understandability.

Conciseness relates with some quality factors, and sometimes conflict with them. Efficiency is one of the quality factors that relates with conciseness. Overly conciseness program used many functions. Developer must include modularity of his program package otherwise he will not gain the full quality of program.

2.3.6 Correctness

Division error to line of code is determined correctness quality factor. Difference between the accuracy and correctness that correctness relates with standard errors.

Correctness relates with the reliability. If software is reliable, it gives correct results in a period of time. Correctness also determines the performance of program. Firstly tester must understand the program for measuring its correctness. If correctness of program is higher then it is understandable and readable. In modern programming the size of code is so higher. If code's size so higher then this code must be modular hence it may not defined as correctness.

2.3.7 Expendability

Expendability increases performance of software. Expanding means that you change program and increase its function's modularity. Firstly program must be readable and simple. If program is so complex then it will be hard to change and expend it.

Before expending developer need to measure the size of program. The size is changed when the program expends. If there are not any problem with the size and data then developer may analyze the function and module of program. Modular program is expended so easily. The changing a module in program is so easier than changing all source codes.

2.3.8 Maintainability

There are many definitions about this process. One of them is following.

"It is a measure of how easily software can be changed because of bugs encountered during operation, user requirements that were not satisfied, changing requirements, and upgrading or 'obsolete' a system . In this view, maintenance encompasses not only the time needed to fix errors, but also time needed to enhance the system's operation. " (U.S. Department of Transportation Federal Aviation Administration, 1991)

Maintainability is part of quality that give us how program can be maintain after developing. It is so hard for developer. This quality factor is determined every life cycle of developing.

Maintainability also relates and conflicts with other quality factors. Complexity conflicts with maintainability. The complex program is harder to understand. Before maintain the program developer and user must understand software. Then it can fix bug. There are many bugs in complex software. Efficiency is also one of factors that conflicts with maintainability. Efficiency is depended on machine system. The developer fixes the bug or writes some module for software but it may not work on requirement system. Maintainability relates with simplicity, portability, adaptability and generality.

2.3.9 Modularity

Improvement of technology is going so fast. Developers and programmers every day write new codes, new programs for these technologies. Modern technologies consist of multifunctional programs. Developer uses module for decreasing the complexity, efficiency and size of program. Modular program helps developer for making program reliable, usable, expendable and etc. Modularity also relates with other factors such as flexibility, portability, reusability

2.3.10 Portability

Portability measures how easily program runs in many different systems. In modern time there are many systems and operating systems that used by user. Portable program is not dependent only operating system and machine, it also dependent from the programming language. Most programming languages are not working on every operating system.

Before writing a program firstly need to select programming language. The operating system must be determined which program will run on it. After this procedure the programming language and operating system are combined. Then written program uses the class of programming language which operating system run the program by using this compiler. Portable program is written following this steps.

2.3.11 Reliability

Reliability is remarkable between quality factors. For that many researcher writes many articles about how increasing the reliability of programs. Reliability is not dependent only program. It also depends on hardware and developer.

In spite of good programmed program, the problem of hardware affects the reliability of software product. For example: The plane's software products must be full reliable.

Reliability factors are related with accuracy. It is depended on developers experience. Good algorithm, using necessary types of variables every time gives you reliable program. Complexity conflict with reliability. There are many bugs and error in complex program and it affects the quality of program. But when developer use the module in complex program the reliability of program increases. Reliable program is also usable, portable, understandable and etc. This quality factors are also related with performance and correctness.

2.3.12 Simplicity

Software program simplicity uses data and control for organizing program and makes it understandable. Simplicity relates with conciseness. This quality factor is proportional to understandability. All programmers understand simple programs code's structures. Modularity also relates with simplicity. Module in the program allows programmer to understand the application. Simplicity is opposite of complexity.

2.3.13 Understandability

Understandability determines how you can understand the program. To be understood the program it must be simple, well commented, good defined variables and coding as standards. Understandability is related to all quality factors. It is conflicted with reliability and performance.

CHAPTER THREE

HALSTEAD'S METRICS AND LOC

In Chapter two software metrics, about its measurements and quality factors are discussed. There are many methods that is used for measurement of software metrics. There are a few methods that used in 70's and 80's years. But nowadays many company and organizations use their own calculation methods. And their measurement covers all development process and all phases of software life cycle.

Measurement of metrics is divided into two parts. They are code based measurement and report based measurement. Code based measurement measures the metrics by using some mathematical calculations. The report based measurement takes all reports from tester and users for determination of quality. In 70's and 80's determination of software quality based on numbers. The methods which were used in that time are McCabe Cyclomatic Complexity Metrics, Halstead's Metrics, RADC's Software Metrics, Albrecht's Function Points Metric, Henry and Kafura's Information Flow Metrics.

3.1 Halstead's Software Metrics

In 70's, Halstead created new measurement methods in Software science. He thinks that without using the code it is possible to calculate some metrics of software. For his method some metrics are gotten from algorithm because it obeys physical law. He used directly, indirectly, statically and dynamically metrics in this method

His calculation based on operators and operands. Operands are the variables and constants. Operators are the symbols that affect the value of operands. For example parentheses are operators. Halstead calculated the software metrics by counting the operands and operators. Before counting operators and operands, code must be written clearly. This counting helps programmer to calculate program length, vocabulary and etc.

Calculating of metrics by using Halstead's method, firstly we must count following parameters.

- $\eta 1$ - number of unique operators.
- $\eta 2$ - number of unique operands
- N1- number of total operators
- N2 - number of total operands.

Let's calculate software metrics by using the Halstead's method

3.1.1 Vocabulary

Calculation of vocabulary is as following.

$$\eta = \eta 1 + \eta 2$$

The unit of vocabulary is defining as word. Word is defined as string and characters in programs. In this equation vocabulary is total of unique operators and operands. Each programming language has own syntactical rules. Same programs give different number of operators in different programs.

Calculate vocabulary in following example.

```
x = y + z;
t = x * x;
```

In this program unique operands are x, y, z and t. Number of unique operands is 4. Operators are =, +, *. In this program return function must be counted as operator. So number of unique operators is 4. In this calculation the vocabulary is 8. Calculate vocabulary of this program is so easy because the code is written so clearly.

The counting method should be established by a programmer who knows the programming language so clearly. Without this knowledge, the counting will be false. And it affects the quality of program.

3.1.2 Implementation Length

Calculation of Implementation length is as follows

$$N = N1 + N2$$

N1 is total number of operands, N2 is total number of operators. These results give total length of programs. In this calculation all operands and operators are counted whether they are unique or not.

Example:

Calculate Implementation Length of following example.

```
x = y + z;
t = x  / y;
```

There are 5 operators and 6 operands in example. The operators are two equal, one plus sign, one division sign and one return functions. The operands are two x, two y, one z and one t. The implementation length is 11 words.

This calculation is easy. Because this measures is linear. But in programming the measure is not line in many time and it is calculated as linear.

3.1.3 Estimated Length

The implementation length can be estimated. Halstead calculated the length as total of operators and operands in the power set. The calculation of this definition is as

$$2^N = \eta 1^{\eta 1} * \eta 2^{\eta 2}$$

Calculation of estimated length is as follow

$$\hat{N} = \log_2 (\eta 1^{\eta 1} * \eta 2^{\eta 2})$$

Differentiated between Implementation and Estimated length the sign is written as $\widehat{N}$. There is a strong relationship between estimated and implementation length. Let's calculates both length parameters. The compare one with other for finding correlation. If this correlation is bigger than 90% then it is well structured programs.

The calculation of estimated length is based on unique operands and operators. If operators and operands will repeats once or two times then it means the estimated length will underestimate implementation length. This length also can overestimate. In the program number of unique operands and operators increases. Then it is not means that the implementation length changes. But the value of estimated changes its value.

3.1.4 Volume

Halstead measures the size of algorithm from the software's parameter. The size of algorithm is important aspect of software. He defines this size as Volume of program. The calculation of Volume is based on implementation length and vocabulary of program. When program is translated from one language to another it changes its size.

The calculation of volume is as follows

$$V = N \log_2 \eta \text{ bits}$$

N is the total number of words, η is total number of unique words in the program.

Halstead used in this calculation combinatory logic. The calculating of minimal number of digits is as $\log_t c$. t is a base of number, c is a number of elements in program. Halstead shows the value of t as 2, because in program all code are stored as binary digits. $\log_2 c$ can represent the each unique words in vocabulary. For example: If our program has 16 unique words so the calculation will be $\log_2 16 = 4$ bits/words. Then Halstead multiplies to this calculation number of total words and at the results it gives us the Volume of program. This measure is calculated normally as follow

$$V = (N1 + N2) \log_2 (\eta 1 + \eta 2)$$

3.1.5 Program Level

Halstead used this metric to measure the ability of the programmer. The lowest program level is 1. It means that there is one statement. The function of this statement is call to a procedure. And this procedure exists outside of the program. The lower degree of program level reduces the program into simplest form.

Project manager and some developers use this metric for calculating the effort of a program. Effort of a program can be measured on every written program. With this measure project managers determine how programmers easily understood and implemented the program again.

If program is written by experienced programmer so the level of program is too high. If the product is coded by group of new programmer then the value of program level is so high. Halstead wrote on his article that lower level is so wordier. Because it is so difficult to understand this type of program and there are many hidden steps in the source code.

Calculation of Program Level is as follow

$$\hat{L} = (\eta\grave{1} / \eta 1) \text{ x } (\eta 2 / N2)$$

$\eta\grave{1}$ is the minimum number of unique operators. The minimum number of unique operators in software programs is given as 2. And Halstead calculated the program level as follow.

$$\hat{L} = (2 / \eta 1) \text{ x } (\eta 2 / N2)$$

3.1.6 Program Difficulty

Program Difficulty is the inverse of Program level. This metric is proportional to Programs Volume. When the value of Volume increases, the difficulty also

increases. The Operators and Operands are repeated will increase the value of Volume and Program Difficulty.

Calculation of Program difficulty is as follow.

$$D = 1 / \hat{L}$$

If we write square roots program in different languages then the rate of program difficulty will change on each languages. Assembler has high difficulty rate among the other programming language.

3.1.7 Intelligent Content

Intelligent content presents the algorithm complexity of programming language. Calculation of this metrics is as follow.

$$I = \hat{L} \times V \text{ bits}$$

$\hat{L}$ is symbol of program level and V is symbol of Volume. The unit of intelligent content is "bit". This metric represents the number of constant information that presents in programming language at any level.

Calculate Intelligent content by using operators and operands. In before equation we used program level and volume. Let's write calculation of this metrics.

$$V = (N1 + N2) \log_2 (\eta 1 + \eta 2)$$

$$\hat{L} = (2 / \eta 1) \times (\eta 2 / N2)$$

Substitute these quantities into the main equation.

$$I = (2 / \eta 1) \times (\eta 2 / N2) \times (N1 + N2) \log_2 (\eta 1 + \eta 2)$$

3.1.8 Programming Effort

Programming effort measures the programmer's ability required to write a program. Calculation of programming effort is as follow.

$$E = V / \hat{L}$$

V is the Volume and L is the Program level. The unit of Effort is a discrimination.

Let's calculate the effort of following program which is written in Basic.

Table 3.1. Basic Code Example

```
Let a = 4.5
input b
let x = b/a*7
print x
```

Let's count the operators and operands that we use for calculations of programming effort.

- $\eta 1$ number of unique operators is 6 (input, "let" and "=", *, /, print, line separator)
- $\eta 2$ number of unique operands is 5 (a, b, x, 4.5 is constant, 7 is constant)
- N1 total number of operators is 9 (two "let" and "= ", input, print, *, /, 3 line separators)
- N2 total number of operands is 8 (two a, two b, two x, constant 4.5, constant 7)

"Let" and “=" symbols are counted together because in Basic" =" is not used without "let".

Calculate the volume of this program.

$$V = (N1 + N2) \log_2 (\eta 1 + \eta 2) = (9 + 8) \log_2 (5 + 6) = 17 \log_2 11 = 58.81$$

Let’s substitute the numbers of Hallstead's parameters in Level equation

$$\hat{L} = (2 / \eta 1) \times (\eta 2 / N2) = (2 / 6) \times (5 / 8) = 10 / 48 = 0.20$$

Calculation effort of this program is as follow

$$E = V / L = 58.81 / 0.20 = 294.05 \text{ discriminations}$$

Effort measures the programmer's mental that how he writes a program by using the operators and operands. If the effort of the programmer is higher, then programmer is fluent in language which program is written and he understands the program very well. Volume is proportional to effort. So large volume requires high level effort.

3.1.9 Programming Time

Programming time is duration of programmers coding. Calculation of programming time is as follow

$$T = E / S$$

S is a Stroud number that changes between 5 and 20. Stroud number is a number of human brains perform effort discriminations per second. Generally in software scientists use Stroud number's value as 18. The programmer who are fluent in coding, his Stroud number is defined as 18.

Let's calculate the time of program in table 3.1 Effort of this program is 294.05 discriminations. Let give the value of Stroud number as 16 discriminations/second. The programming time is

$$T = E / S = 294.05 / 16 = 18.3 \text{ seconds}$$

For Hallstead the programming time is amount of time that how much time programmers understand the program. So with this calculation you also can calculate future application. But firstly we must calculate the effort of programs as follow.

$$E = V^3 / \lambda^2$$

V is volume, λ is Language level. So from this equation we can calculate the future programming time.

3.1.10 Language Level

Calculation of language level is as follow.

$$\lambda = L^2 x\ V$$

L is program level, V is volume. Language level is metrics that gives us how language represents algorithm. Let's calculate language level of program in Table 3.1. Substitute Volume and program Level into this equation. And language level is

$$\lambda = (0.20)^2 \text{ x } 58.81 = 2.35$$

This amount allows project manager and programmers to select programming language for their projects. The project developers write their code in different language and measure their language's level. Then higher level of programming language is chosen for implementation

3.1.11 Number of Bugs

Calculation of number of bugs in program is as follow.

$$B = V/ E_0$$

V is number of Volume, $1/ E_0$ is average number of discriminations which people make for a bug during the coding. E_0 is determined previous work of developers, but Halstead determines it as 3200. Let's calculate the number of bug of program in table 3.1

$$B = V / E_0 = 58.81 / 3200 = 0.01 \text{ bugs}$$

Number of bugs in this program is less than 1 because it has only few statements. In generally, the number of bug is higher when the volume of program is large. The number of bugs in real life program is more than example.

3.2 Line of Code (LOC)

First metrics in software science is Line of Code. This metrics first time used in 1960 year. Programming cost, defects rate and productivity of the program can be calculated by this metrics. Economic of software is measured by using "dollar per LOC". Productivity of software is measured by "LOC per time" and defects rate is measured by "defects per KLOC". KLOC is 1000 LOC.

Determination volume of code by LOC can be compared with other programming language. Program gives different LOC in different languages. First time when the LOC was used, there was only assembly language. So count the LOC and calculation of program efforts was so easy.

What is the LOC? Line of code is a number of codes that is not a comment and blank. Code is combination of headers, functions and statements. There are two types of counting method in LOC. Physical and logical line of code.

Physical LOC is code that contains inside the number of comments and blanks line. Before measuring the Line of Code by using this method you must ask yourself what you are measuring and what this measure will give you. If you want to measure the productivity, complexity of program then you can use for measuring the executable physical line methods. What is executable physical line? Inside of this line there are not any comments and blanks line. This type of measurement is so easy than logical if programming language standardized statements in one format. In modern language some of developers creates own class for substitution of some functions. And in counting time the organizations determine this class name as blank. Before testing developer must introduce its own class to tester and input it as class into the counting program.

Logical LOC counts the statements and functions line in program. Logical LOC is independent physical form that counts logic statements in code. Logical LOC is used for determination effort, cost, design, coded and etc. Advantages of this source statement are not dependent on programming style. And changing of logical LOC is not affected the source code and its formatter. Disadvantages of this types LOC is

counting rule determined by organizations. So functions, modules of statements are delimiter from language to language. It means that you cannot compare this measurement with other code that writes in different language. But if delimiter is not defines as explicitly so they determine own rules or use general rules. There are many languages that have more one type delimiter. Pascal, Basic C has own comment counting rules. The coding rule is different in many languages. Many languages need to use semicolon at the end of line, but some of them do not need to put this type symbol. So developer must write the code by the rule that is determined by organizations. And it makes easy counting logical source statements determined number of Logical LOC

Comments are a string that has not any effect to program's code. Writing and deleting code is not change structure of codes. Comments are physical lines that many time counts by some testers and project managers. Comments are divided into many types by its purposes. Header comments are the explanation of class or programs. Explanation before procedure and module is call pre-module comments.

Blank Lines is a physical line that is not defined by any symbol. Blank lines effects many quality factors such as readability and maintainability.

The advantages of LOC are being simple for measurement. The disadvantages of Loc is that it is depended on languages, bad structural codes give excessive line code and cannot easy understand it.

Some of tester and scientist think that they measure the productivity of program by LOC. But modern programming is different than in the 60's. In 60's years developers did not used functions, procedures and class in their code. But now there are many different languages that is used by developers. The complexity of program affects the productivity of product. So in modern programming the complexity program nearly takes 100 line codes. And this type of program is same as 10000 line code in 60's years. I think we cannot use this metrics for measuring of productivity in modern programming. Because the productivity of program which has 20 line of codes is less productivity than system that has 1000 line of code. But first program is

more effectiveness than others and it used many functions and class in coding. Line of code is a product metrics.

3.3 Software Quality Factor Based On LOC and Halstead Metrics

Software Quality is determined by software metrics. There are many measurement methods for determining metrics and quality factors. Some Halstead's metrics relate with quality factors.

3.3.1 Implementation Length

Implementation length is determined by calculation of operators and operands. This counting is correlated with the complexity. And only the number of unique operands correlated with the modularity. Increasing implementation length also affects to the complexity. Complexity and this quality factor depend on size of program. When the size of program is increasing it is difficult to understand it. So Halstead's implementation length is used for determining Complexity of program.

3.3.2 Volume

Volume determines the size of program or algorithm by bits. The running of big volume program is depended on system requirement. But some tester organizations use the volume of program for determining conciseness and efficiency. Decreasing the volume of program increase only the running speed of product.

3.3.3 Program Level

In software engineering this metrics determines how programmers understand the program before writing. Many project managers use this metric determine the effort of program and ability of programmers. Program level is related with program volume. Halstead noted on his research that lower level of programs is wordier. So the program level is proportional to understandability of program and it is difficult understand the lowest level program.

3.3.4 Difficulty

Difficulty is inverse of program level. Difficulty is not proportional to quality factor that are determined by program level. Complexities of program are measured by this metric. Complexity is divided into two parts structural and psychological.

3.3.5 Intelligence Content

New developers write same algorithm in different language for knowing the difference of intelligence content between two programs. At the result the intelligence content of programs are nearly same with others and it took 10 percent of average. So in 1960 years the tester determined that this metric measures the complexity of program. But for many organizations this result is not true because it is written by new developer not experienced programmer. When the programmer is written by professional programmers so the intelligence content will not give same result. For experts standpoint, the Intelligence content relates with efficiency of program because it is an estimated of volume.

3.3.6 Programming Effort

Programming effort is used by many researchers as different version. Sheppard, Milliman and Curtis used this metric for determining of program understandability. For these researchers the effort is proportional to understandability.

According to the research effort is related with programming development and program clarity. Effort determines the modularity of program according to Baker and Zweben's research.

Harrison used the equation of effort for correlation between effort and module. By this correlation he found maintenance time that delivered by module. In Study, module is defined as software component. But in experience it is subprograms. He used effort for finding the bug of module and determining the maintenance of program.

3.3.7 Number of Bugs

According to equation Volume is proportional to number of bugs. The program is written as same relative quality by many programmers. And at the result we determine that Volume and number of bug are not linear. Number of Bugs relates with correctness and maintainability.

Halstead's software metric relate with the following quality factors

- Complexity
- Conciseness
- Correctness
- Expendability
- Efficiency
- Maintainability
- Modularity
- Readability
- Simplicity
- Testability
- Understandability

Determination of software quality factors by Halstead's metrics are not used in modern quality determination. Implementation length relates with flexibility, reusability, portability and etc. For some tester this metric relates with understandability. Because this quality factor is complementary to complexity.

Volume and Programming efforts also relates with many quality factors such as complexity, modifiability, modularity, number of bugs and etc.

Halstead's metric is derived from programming efforts, volume and implementation length. Some quality factors are determined by documentation reports. But Halstead metrics are determined by mathematical calculation. For this reason Hallstead metrics are not correlated all quality factors.

The other type of metrics that related with quality factors is line of code. Above line of code and its functions are discussed. By line of code the size of programs, its defects rate and productivity can be measured.

If the code size is calculated as physical lines then complexity and volume of code should be determined. The measurement of LOC is divided into many parts such as CLOC, BLOC, SLOC and etc. CLOC is a comment line of code, BLOC is a blank line of code and SLOC is a Source line of code. CLOC measures the number of comments in the program's source code. Comment is used for explanations some modules or functions in the code. Number of comments is increasing the understandability of program, but more comment is not more effectiveness. Blank is a line that there is no any symbol in it. The number of blank determines the inaccuracy of programmers. It is also determine complexity and maintenance of program. More blanks line is not proportional to clarity, accuracy and understandability. Source line code is determined by logical line. There are only statements in this line. This metric determines size of code. By this metric we can measure volume, complexity of program and ability of programmers.

CHAPTER FOUR

DEVELOPING A TOOL

The determination quality of software program is a important for product's companies since the first software program was written. Improving technology and programming languages also affects for analyzing of products. Programming style is changing years by years. First programming type is linear programming. Then programming is changed to object orienting programming, file programming, database programming and etc. Increasing programming style affects to the complexity of programs. Creating new measurements method is also depended on developing program style and programming language. So in modern programming there are many analyzing style that are used for prescribing quality of products. Each programming language has its own analyzing method.

Generally, user wants to know the quality of programs. But companies want to know quality of programmers before the programs. Because the qualitative programmers create quality program. So companies start measure the quality of programmers by the near of product. The quality of developers is determined by his own code's parameter.

Halstead's and Line of Code metrics are used for analyzing software and developer's work in this thesis. Halstead's metrics first time was used in 70's years. The programming type was linear when Halstead's method was used. But now most useful programming type is object oriented. For calculation metrics of programs which is written in this type is difficult. Therefore we cannot apply Halstead method's to this calculation metric.

Line of Code metrics first time was used in 60's years. It was easy to calculate number of line in programs. programmers have started to use more than one programming languages in one software product since 80's and it was difficult count the line of code as accuracy. Testers started to use function as line of statement

which relates with other language. In these periods the productivity of software increased but number of LOC decreased. After 90's the number of programming language increased. And programmers start to use more than 4 programming languages in one product. So it was difficult make a standards calculation for all programming language and each tester starts to create its own calculation method. Increasing number of programming languages changes the measurement type of LOC. Some tester used physical line, some tester logical statement line. In large projects there are many non-code workers such as project manager, designer, quality assurance, data base administrators and etc. There is one problem that none of this people can measure Line of Code. For measurement of Line of Code firstly tester must know the programming language. As of 2013, software industry has 2500 programming language but 2400 of them are becoming dead languages. And other remainder languages have its own measurement standards. Many organizations use own standards for measurement.

The main problem of modern programming that without seeing the source code organizations cannot determine the quality of program. And the programmer does not want to give the source code to anyone. Company begins to test its own program. At the result it consider that this program quality or not. Now there are some organization has own quality standards such as ISO, IBM and etc.

4.1 Creating a Tool for Measuring the Program Source Code

In this thesis for determination of the program quality without seeing source code the web tool is created. Halstead’s and LOC metrics are used in this measurement. It is known that applying these methods for modern programming is difficult. Firstly web tool measures the LOC of source code on the internet. This tool measures the CLOC, BLOC and SLOC of program. At the result number of comment's line, blanks line and statements line are inserted into the database table. After this programmers enter into the database the unique and total numbers of operator and operands. So tester cannot see the source code of program.

How is developing tools working? For collecting and determining Halstead's and LOC metrics of source code windows 7 home premium are used as server of web

tool. Use the apache as internal server and MYSQL 5 as database. This tool is coded in 4 programming language. They are PHP 5, Java, C# and JavaScript. Firstly Programmer must enter to special IP address and make registration for entering his programs source code data. Then his source code metric are analyzed by web tools. LOC metric tool is written in java and C#. It determines comment, blank and source lines. Programmer must enter Halstead's metric into the tool by his own hand.

This tool measures the LOC of program which is written in 45 programming languages. Before calculating the LOC programmer must do the following process

- If you are using the linear programming you must write the function code under the function's name in the main code.
- If you are using the object orienting programming you must enter class code which you write in the main code.
- If you are using the class of library then this class will count as one source line.
- If you use the two programming language in one program then shows the type of file as which language take main.

In this tool we put all 45 languages specific symbol in the database and write the standards of each programming languages. Programmer enters its source code into the text pad and tool calculates the physical and logical line of program on internet. Then number of comment lines, blank lines and source lines automatically enter into database table.

After calculation of LOC programmer must enter the Halstead's metric into database. The number of operators and operands will be counted by programmers. Because there are many problems for determination of the operators and operands. In modern programming there are some words that used as operator and operands. Programs may make a mistake the determination of these type operators. So some standards must be defined for this calculation. Before measurement programmers must do attention in following calculation rules.

- If you are writing linear programming and using the function in code then you must count all operators and operands of function. And you must enter the sum of all this number into the database
- If you are writing object orienting programming and using the class in code then you must count all operators and operands of class. And you must enter the sum of all this number into the database.
- If you are using the class which you write then you must count this class's operators and operands.
- If you are using the class from program library then you must count it as operator.
- If you are using the two or more programming language in one program then you will enter the sum of all operators and operands of two language's source code.
- If there are any operators and operands as same name then you will take them as operators and operands by its position.

Before counting, programmer must know what he counts. In modern programming operator is argument that defined statement. There are some words has multiple part that occur operator set. They define as one compound operator. There are some words that represent in one function and this function is counted as one unique operator. But some words have different functions so these are counted as two distinct operators. For example: if ¥ or ^ forms are denoted in source code then these two symbols are represent as one function. If * symbol is defined as comment and multiplication sign in the code then this symbol must be counted as two unique operators. The operands are mostly variables and constant in source code. There are some operands that are written in different form but counted as one unique operands. For example if variable name denoted by velocity or vel then this two form represent as one. Conversely, the one variable may be counted as two unique operands. For example: If constant 5 is given as number of array and also it denotes length of something as 5m, then this constant is counted as two arguments.

The measurement of operators and operands is not monotonic and repeatable. Because programmers may count the word that is not a operand or operator. And it

increases the value of measurements. So different reader define different operator sets. A software tool's measurement is also non monotonic. Although it has a good define standards for identifying operator and operands, it also makes mistakes in counting common operands.

4.2 Analysis the Parameter of Source Code

The purpose of this project is analyzing modern program's quality without seeing source code. There are many programming style in modern time. Many quality analyzers make attention to the programming style. But quality is not depended on programming style. It depends on structure of program. Quality is divided into 4 parts. They are product quality, manufacturing quality, user perceived quality and economic quality. User satisfaction affects all these quality types.

Software must do what the customers want in it to do. This term is first mission of programmers. So it means that user makes a important role in software production and quality. Before writing the code of program project managers or developers must collect the request of user. User's view of the quality is called quality in use. The behavior and work of developer is measured by work system. The output of work system measured as effectiveness, productivity and user satisfaction. User data satisfaction is measured by 5 point scale. They are very satisfied, satisfied, neutral, dissatisfied and very dissatisfied. The quality is determined by these scales and based on design metric, system requirement, density of failure and etc. We can determine the capability, functionality, usability, performance, install ability, documentation and overall of product by user satisfaction metrics. In this project my tools cannot determine this quality types. For this determination program is needed to run firstly then opinion of user should be collected. Web tool determines the product quality.

95 programmers' sources codes have been collected in two month. Some of unfit data is bolted from collection data. Then web tool starts calculating the Halstead's metric of data and insert it into new database table. The calculation is in Table 4.1 and Table 4.2

Table 4.1 Calculation of Halstead's metric of collection data

```
$plen=$topnd+$toprt;
$voca=$uopnd+$uoprt;
$pvol=$plen*log($voca,2);
$prlen=($uopnd*log($uopnd,2))+($uoprt*log($uoprt,2));
$lev=(2*$uopnd)/($uoprt*$topnd);
$diff=1/$lev;
$effort=$pvol/$lev;
$time=$effort/15;
$error=$pvol/3200;
$icont=$pvol*$lev;
$llev=$pvol*$icont
```

Table. 4.2 Insert Halstead's metric into table

```
INSERT INTO yap (`yap_id`, `pro_id`, `prog_length`, `vocab`, `prog_vol`, `pred_length`, `level`, `difficulty`, `effort`, `time`, `error`, `intel_cont`, `lang_lev`) VALUES
('0','".$pid."','".$plen."','".$voca."','".$pvol."','".$prlen."','".$lev."','".$diff."','".$effort."','".$time."','".$error."', '".$icont."', '".$llev."')";
```

Vocabulary - This module defines number of unique words in the source code. This module must be well defined. The unique operators and operands must be counted once time. This metrics applies the implementation of algorithm. If algorithm is less than one operand and two operators then it is undefined. Metrics produced integer number. The unit of metric is word. This module relates with clarity, complexity, simplicity and understandability Calculation in our thesis is as follow

$voca=$uopnd+$uoprt;

$voca is vocabulary, $uopnd is unique number of operand and $uoprt is unique number of operator.

Implementation length - This module defines the total number of words in the program. This code also applies the implementation of algorithm. Implementation length is not monotonic. This metric produce the integer number. The unit of metric is words. It relates with clarity, complexity, simplicity, performance and understandability. Calculation of implementation length in our thesis is as follow

$plen=$topnd+$toprt;

$plen is implementation length, $topnd is total number of operands and $toprt is total number of operators.

Estimated Length - estimate the total number of operands and operators in program. It is also same as vocabulary in applying method. This metric produces a real number. The difference of estimated length between implementation lengths is the equation is nonlinear. The unit of metric is words. It relates with clarity, complexity, performance, understandability and simplicity. Calculation of this metric in our thesis is as follow

$prlen=($uopnd*log($uopnd,2))+($uoprt*log($uopnd,2));

$prlen is estimared length, $uopnd is a number of unique operands and $uoprt is a number of unique operators.

Volume - This metric calculate the binary digits of all program's function. Volume is used in comparison size of program in different languages. Metric produces a real number. And the value of metric must be higher than and equal to 4.755. If the volume of large algorithm is less than small algorithm so it can be implemented so tersely. The unit of metric is bits. It relates with clarity, performance, reliability, complexity, simplicity and understandability. Calculation of volume in this thesis is as follow

$pvol=$plen*log($voca,2);

$pvol is a volume, $plen is implementation length and $voca is vocabulary of program.

Program level - measure the implementation of module in source code. When the number of operator is two and number of operand is one, this program's level of source code is higher. Metric produces a real number. The value of this metric is less than and equal to one. This metric is unitless. Program level relates with clarity, understandability, simplicity, complexity and performance. Calculation of metric in our thesis is as follow

$lev=(2*$uopnd)/($uoprt*$topnd);

$lev is a program level, $uopnd is a number of unique operands, $uoprt is a number of unique operators and $topnd is a total number of unique operands.

Intelligence Content - represent quantity of information in any function, in any language, at any level. This metric produces real number and it value is bigger than and equal to 4.755. This metric measure the size of algorithm that based on implementation. The unit of metric is bit Intelligence content also measures the impurity of algorithm implementation. It relates with conciseness and efficiency. Calculation metrics in thesis is as follow.

$icont=$pvol*$lev;

$icont is a intelligence content, $pvol is a volume and $lev is a program level of program.

Programming Effort - measure the number of discriminations when programmer converts the algorithm to the programming languages which he knows fluently. Programmer selects the vocabulary words and uses it in N times in implementation. Each mental discrimination in selection process defines the effort to difficulty understanding. This metrics also produces a real number and its value is higher than and equal to 4.755. The unit of metric is discrimination. This metric relates with clarity, complexity, reliability and simplicity. The calculation of metric in my thesis is as follow.

$effort=$pvol/$lev;

$effort is value of effort. $pvol is volume and $lev is program level of program.

Programming time - measure the time that programmer expend for writing program's source code. This metric produces a real number and its value is greater than and equal to quarter of the second. Most problems in calculation of programming time is determination of Stroud number. Programming time is no monotonic and repeatable. Because the result programming time may be shorter for the program has high programming effort than for a program is lesser value of programming effort. The unit of this metric is second. It relates with performance. Calculation of programming time in this thesis is as follow.

$time=$effort/15;

$time is a programming time and $ effort is programming effort of program. In this research the Stroud number is taken as 15. In many Halstead's calculation the Stroud number takes as 15, it is general Stroud number of normal programmers.

Language Level - This metric measure the efficiency of language then algorithm is implemented in. This number produces a real number which is less than or equal to the volume of program. It does not relates with other quality factors. Because it measures the language level and it does not affect the quality of program. Calculation of language level in our thesis is as follow.

$llev=$lev*$icont;

$llev is language level. $lev is a program level and $icont is intelligence content of program.

Number of Bugs - measures the number of bugs of program that occur in real measurement time. The metric produces is positive real number. Most problem is in determination of E_0 value. It is not monotonic and repeatable. Because programmers may make a minimum bug in high volume program than other. The concentration of programmer may changes day by day. It also depends on programming language. If programmer knows the language as fluently so the number of bugs and E_0 will be

minimum. The unit of this metric is bug. Number of bugs relates with maintainability. Calculation of number bugs in this thesis is as follow.

$error=$pvol/3200;

$error is number of bugs and $pvol is a volume of program. E_0 is defined as 3200. In many calculation of Halstead's metric Halstead determined the typical value of discriminations per bug as 3200.

Difficulty - This metrics is inverse of program level. This metric is also unitless. Metric produces a real number. It relates with complexity, simplicity and clarity. Calculation of difficulty in our thesis is as follow.

$diff=1/$lev;

$diff is difficulty and $lev is program level of program.

4.3 Determination of Program and Developers Quality

The quality metrics is used for measurements for software quality factors. There are many quality factors in software science but in this thesis we will measure the quality factors that are determinate by Halstead's and LOC metrics. Some of metrics are applied broadly, and some of them are not related with quality factors.

The source codes are written in 7 different programming languages by programmers from 36 countries. In this thesis I calculate the quality factors per country and programming languages. 318 source codes are registered in our project. The useless data is automatically illuminated.

The number of programmers is grouped by programming languages and countries in Table 4.3 and Table 4.4

Table 4.3 Number of programmers by programming languages.

Language	Number of programmers
C	9
C#	8
C++	10
Java	29
Lisp	5
Pascal	8
PHP	26

Table 4.4 Number of programmers by countries.

Country	Number		Country	Number
Azerbaijan	8		Niger	1
Belgium	1		Pakistan	7
Botswana	1		Philippines	6
Brazil	2		Poland	1
Bulgaria	1		Romania	1
Cameroon	1		Saudi Arabia	1
Egypt	7		Serbia and Montenegro	1
Ethiopia	1		Slovenia	1
Ghana	1		South Africa	2
Honduras	1		Sri Lanka	4
India	10		Taiwan	2
Indonesia	1		Tanzania	1
Iran	3		Thailand	1
Iraq	2		Tunisia	6
Italy	2		Turkey	1
Kenya	2		United Arab Emirates	2
Macedonia	2		United Kingdom	1
Malaysia	4		USA	6

4.3.1 Clarity

This quality factor determines the metric that how people understands the program. How programmers understand the source code. How user understands the interface of program. This factor applies all programming source code. Normally program should be coded with clarity. Because many programmers writes code, reviews it then code some more. This is a programmer based factor not user oriented.

We determine this quality factor with Halstead's effort. Effort is also used for measurement of programmers ability that how he understand the program. It is function is same as clarity. So effort determines the software clarity.

Clarity of programs by programming language also gives us the writing complexity. The minimum value of clarity means that it is difficulty writing code in this programming language. The clarity of programs by programming languages is as Figure 4.1. And the clarity of program by countries is as Figure 4.2

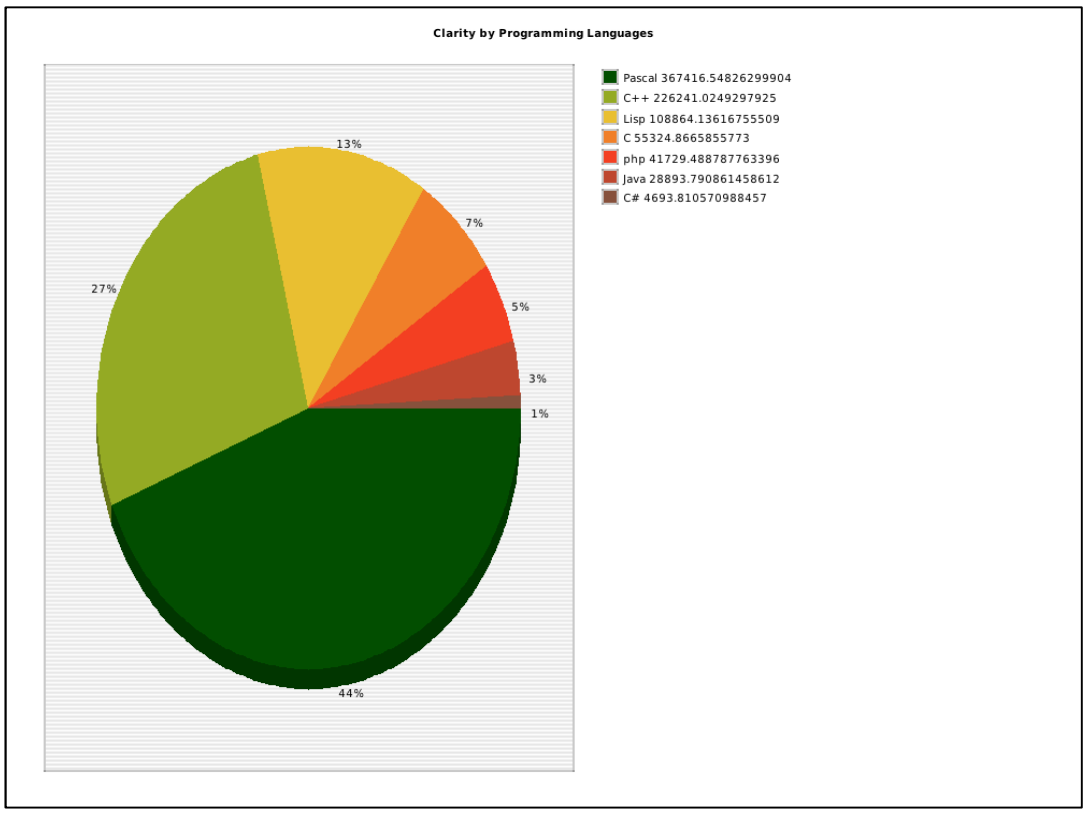

Figure 4.1 Clarity of programs by programming languages

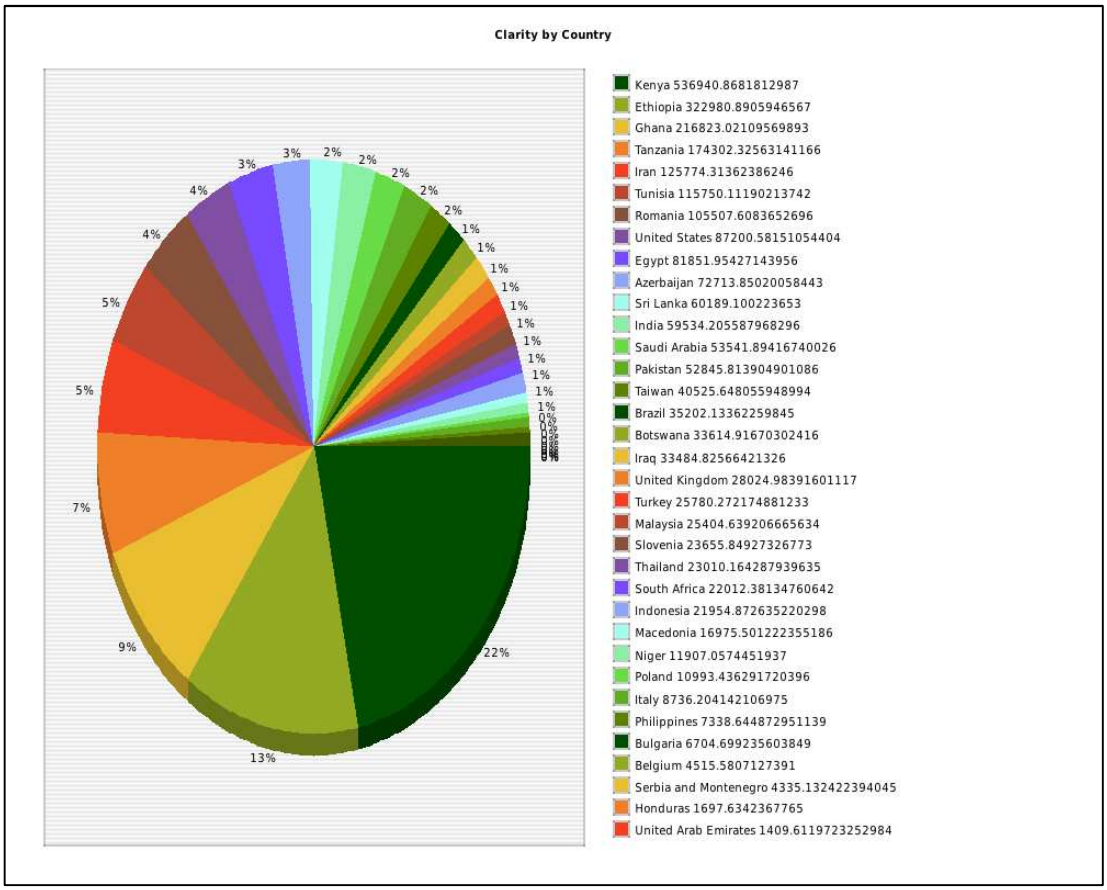

Figure 4.2 Clarity of programs by countries

4.3.2 Complexity

There are many types of complexity. Such as structural complexity, psychological complexity, design complexity and etc. In this thesis only structural and psychological complexity are determined. Other quality factors are based on user satisfaction. The two main sub factors apply to all program's source code. These two quality factors are programmer orienting, not user orienting. These two sub factors are inverse of simplicity. But design complexity is inverse of usability.

Structural Complexity - is complexity of source code. It means that how many operators and operands are used by programmers. So the total number of operators and operands determine structural complexity of program. So the sum of total number of operators and operands give us Halstead's implementation length. The

structural complexity of programs by programming languages is as Figure 4.3. The structural complexity of programs by countries is as Figure 4.4.

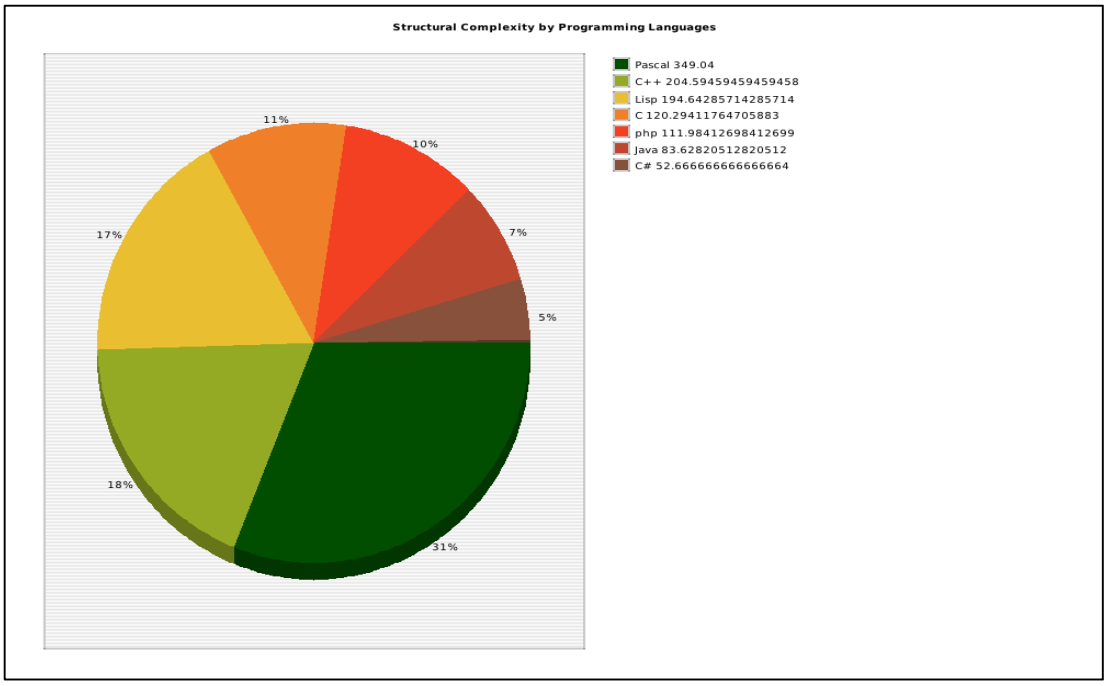

Figure 4.3 Structural complexity of programs by programming languages

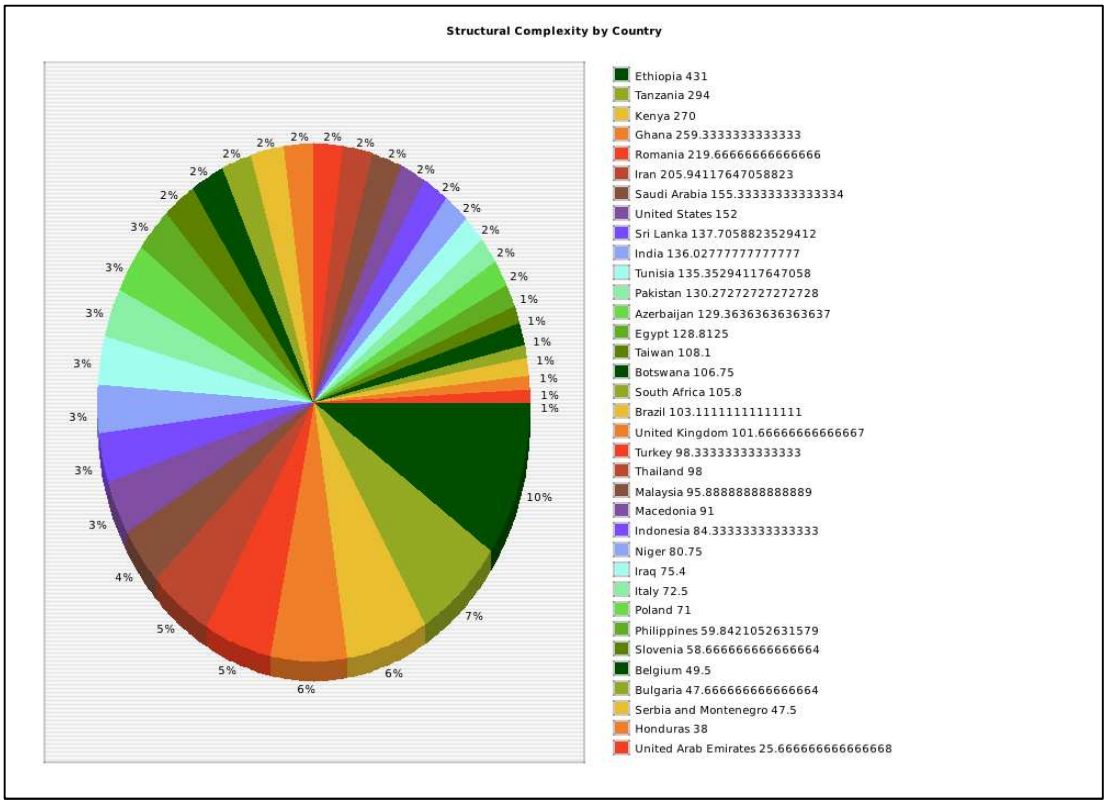

Figure 4.4 Structural complexity of programs by countries

Psychological Complexity - determines how programmers understand the program. This complexity is not defined as single dimensions. Sometimes many testers measure this quality factor by Halstead's effort. The 60% of effort relates with this quality factors. But Halstead's difficult is 100% relates with complexity between Halstead's metrics. This quality factor also perceives the structural complexity, programmer characteristics and problem complexity. The complexity is depended on program's source code, not programmer's ability. This complexity types is also measured programming behavior of developers. The Psychological complexity of programs by programming languages is as Figure 4.5. The Psychological complexity of programs by countries is as Figure 4.6.

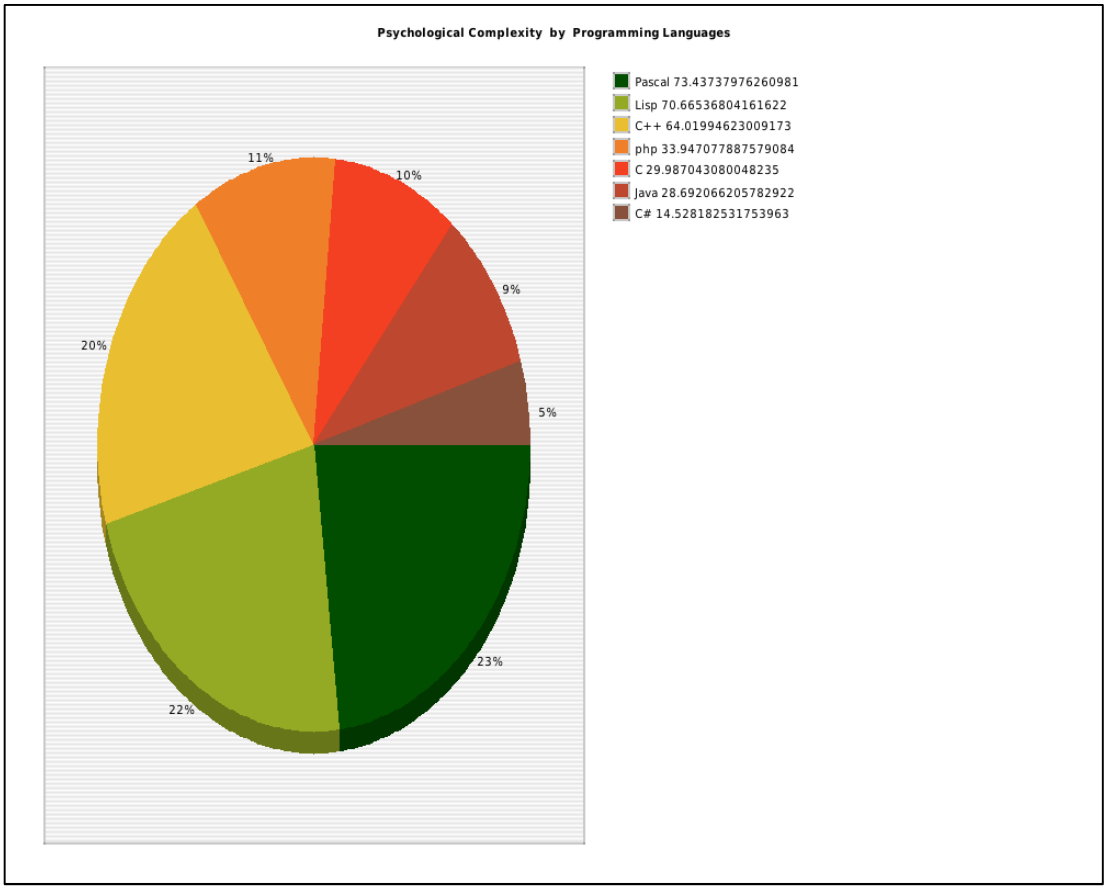

Figure 4.5 Psychological complexity of programs by programming languages

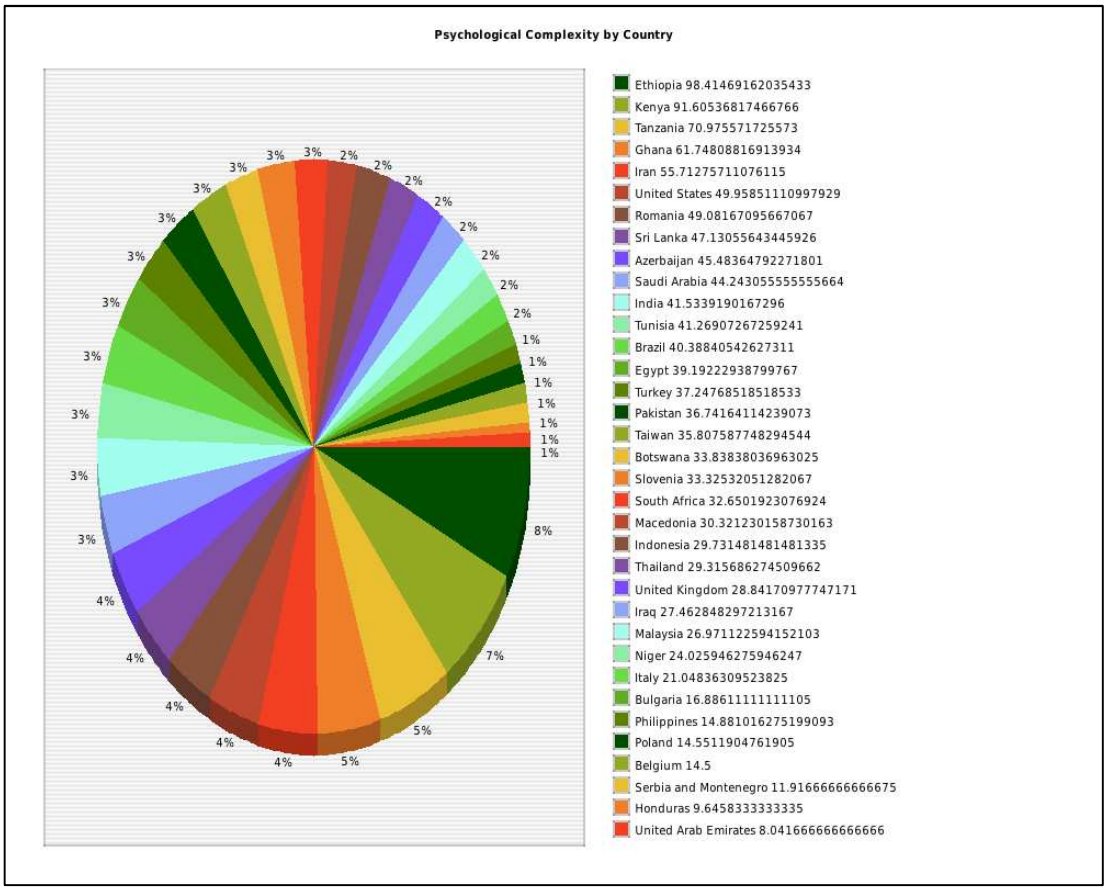

Figure 4.6 Psychological complexity of programs by countries

4.3.3 Conciseness

This quality factor measure the program's code that there are not any extraneous information. Conciseness applies to the source code which the functional necessity is explicitly stated. This quality factor is also programmer-oriented, not user oriented.

The intelligence content metrics relates with this quality factor. This metric is also determine the efficiency of source code. Other name of efficiency is effectiveness. Conciseness is also encompass with efficiency. The value of Volume affects to the intelligence content and it also affects this quality factor. The Conciseness or efficiency of programs by programming languages is as Figure 4.7. The Conciseness or efficiency of programs by countries is as Figure 4.8.

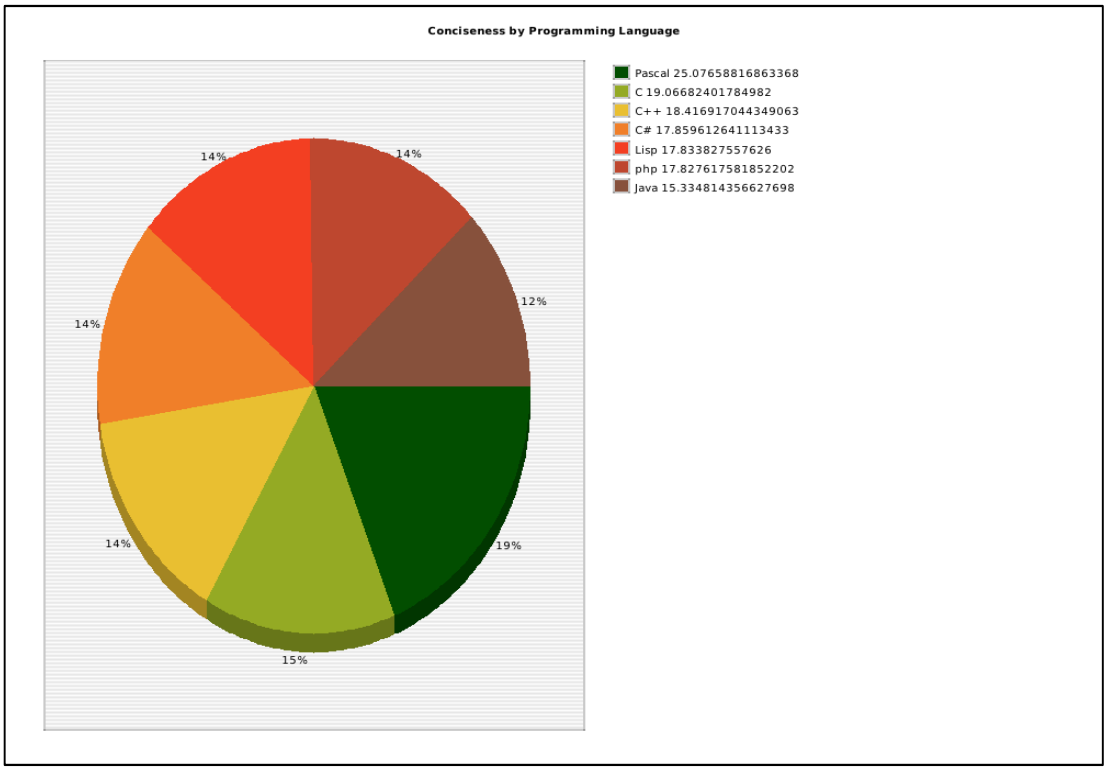

Figure 4.7 Conciseness or efficiency of programs by programming languages

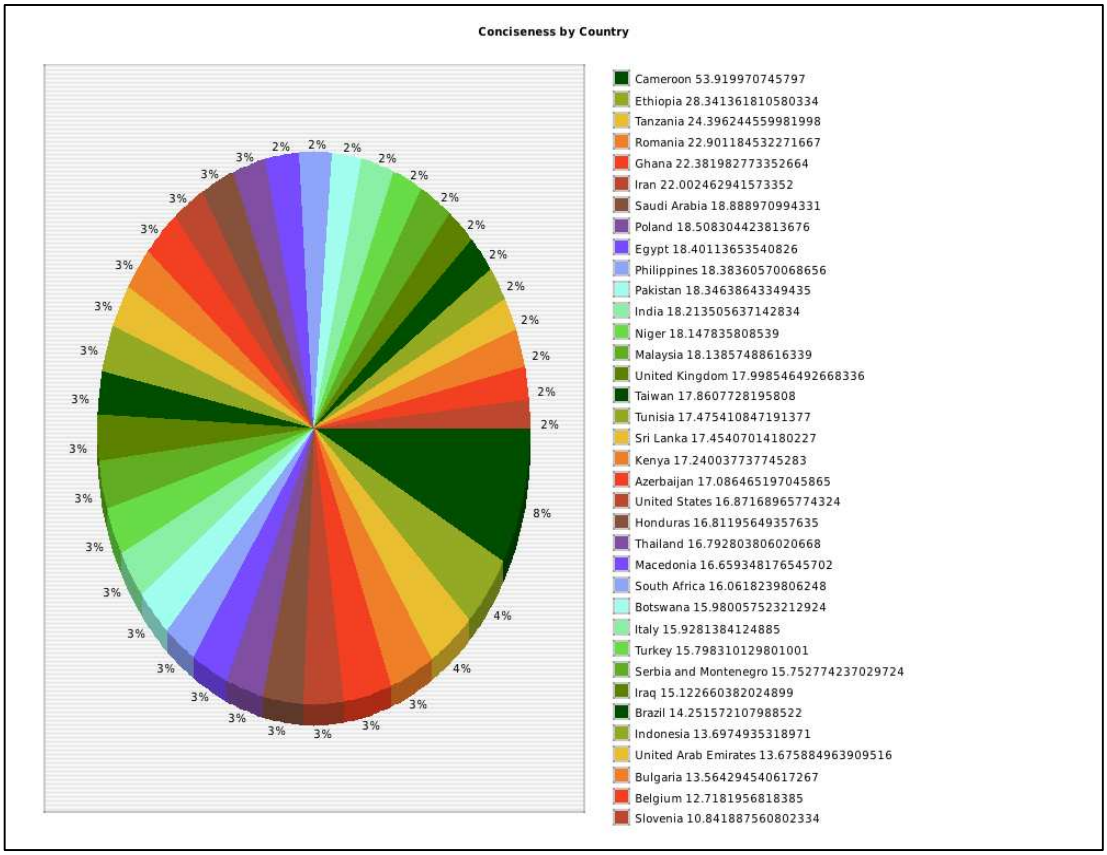

Figure 4.8 Conciseness or efficiency of programs by countries

4.3.4 Correctness

This quality factor measures density of error in source code. Correctness applies to the source code which the functional necessity is explicitly stated. This quality is same as others. It is also programmer oriented, not user oriented. Before measuring the correctness the completeness must be satisfied.

Determination of this quality factor is based on SLOC and Halsted's bug number. Firstly the bug number of source code is determined. Then results is divided SLOC of program. We must calculate this quality factor as this method. Because less bug number is not means that it is correctness. For determination of factors I should calculate by this method. In this graph minimum value gives high correctness. The incorrectness of programs by programming languages is as Figure 4.9. The incorrectness of programs by countries is as Figure 4.10.

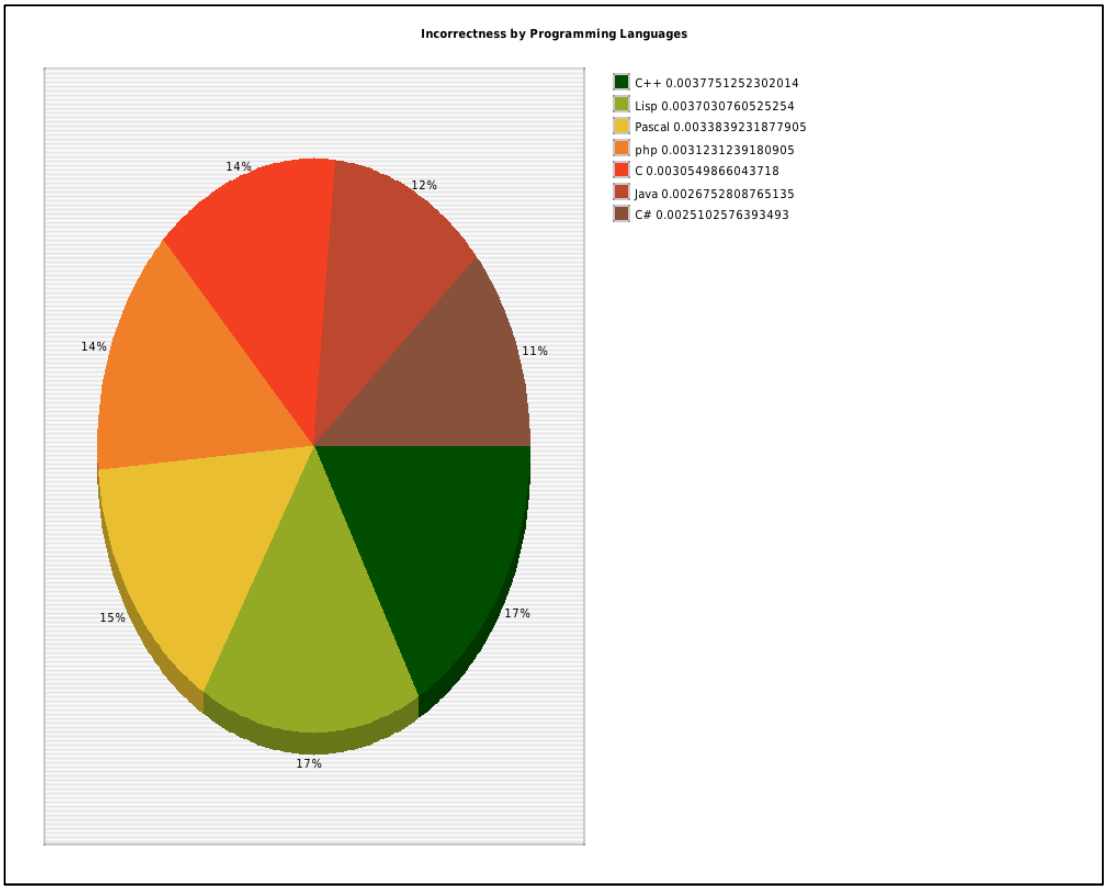

Figure 4.9 Incorrectness of programs by programming languages

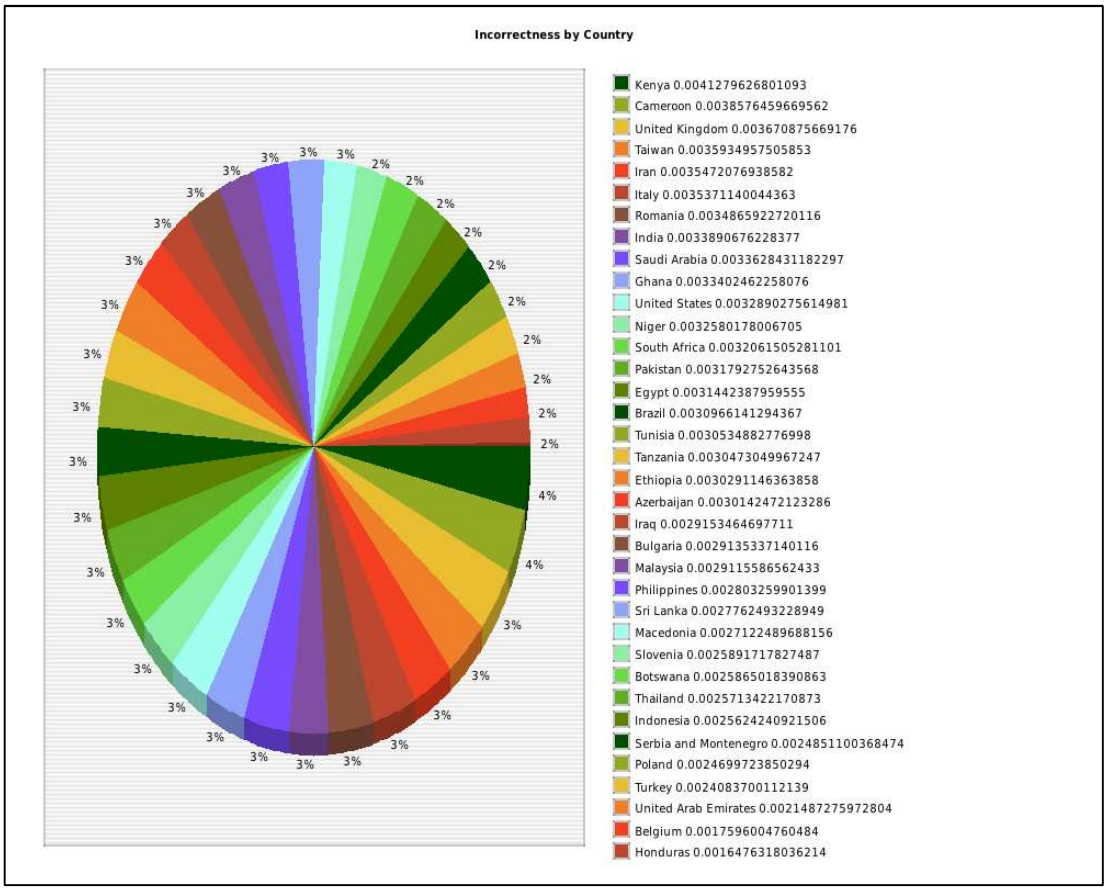

Figure 4.10 Incorrectness of programs by countries

4.3.5 Maintainability

This quality factor measures how easily programs are fixed. Maintainability applies to the source code which has a failure to fix. This quality factors is not confused with portability, reusability, flexibility and etc. None of this quality factors affect to change source code failures.

The number of bugs in Halstead's method relates with the maintainability. Then high number of bugs is gives as minimum maintainability of software. It was so difficult and takes more time to fix the failure. And this calculation is makes maintainability independent form understandability and complexity. But in real life more complexity program has more failure and it means that more complexity program has less maintainability. The less number of bug is more maintenance. The maintainability of programs by programming languages is as Figure 4.11. The maintainability of programs by countries is as Figure 4.12.

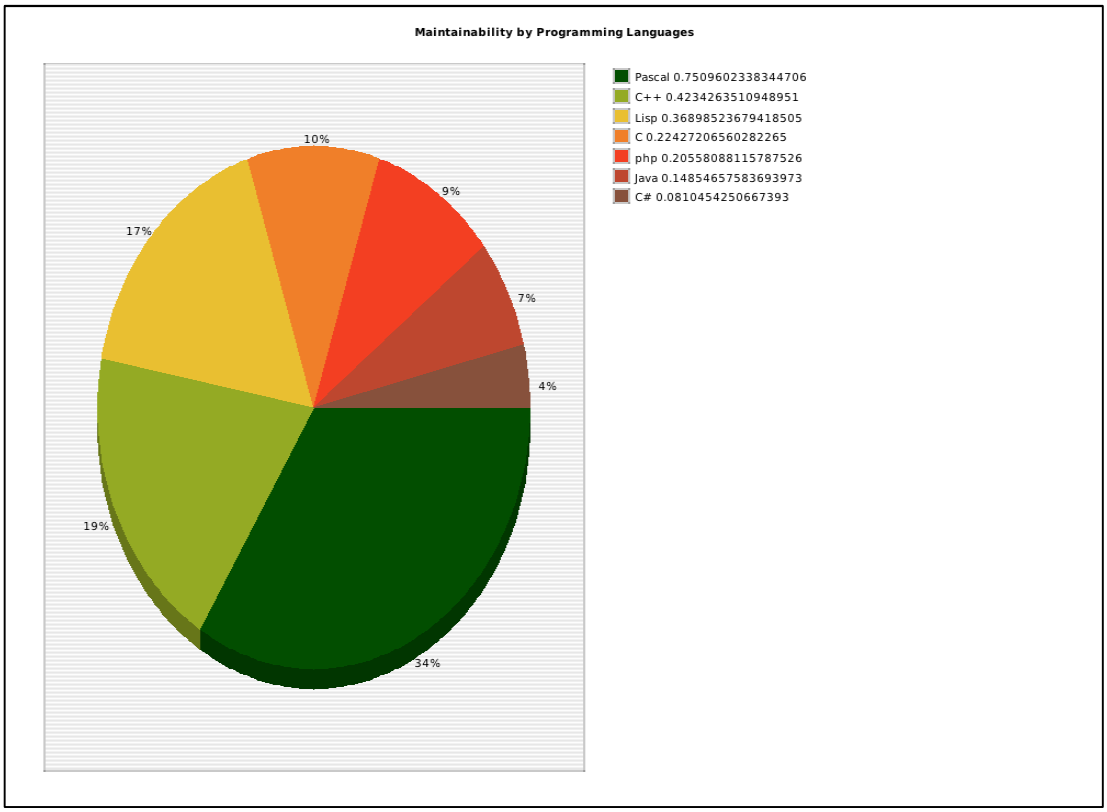

Figure 4.11 Maintainability of programs by programming languages

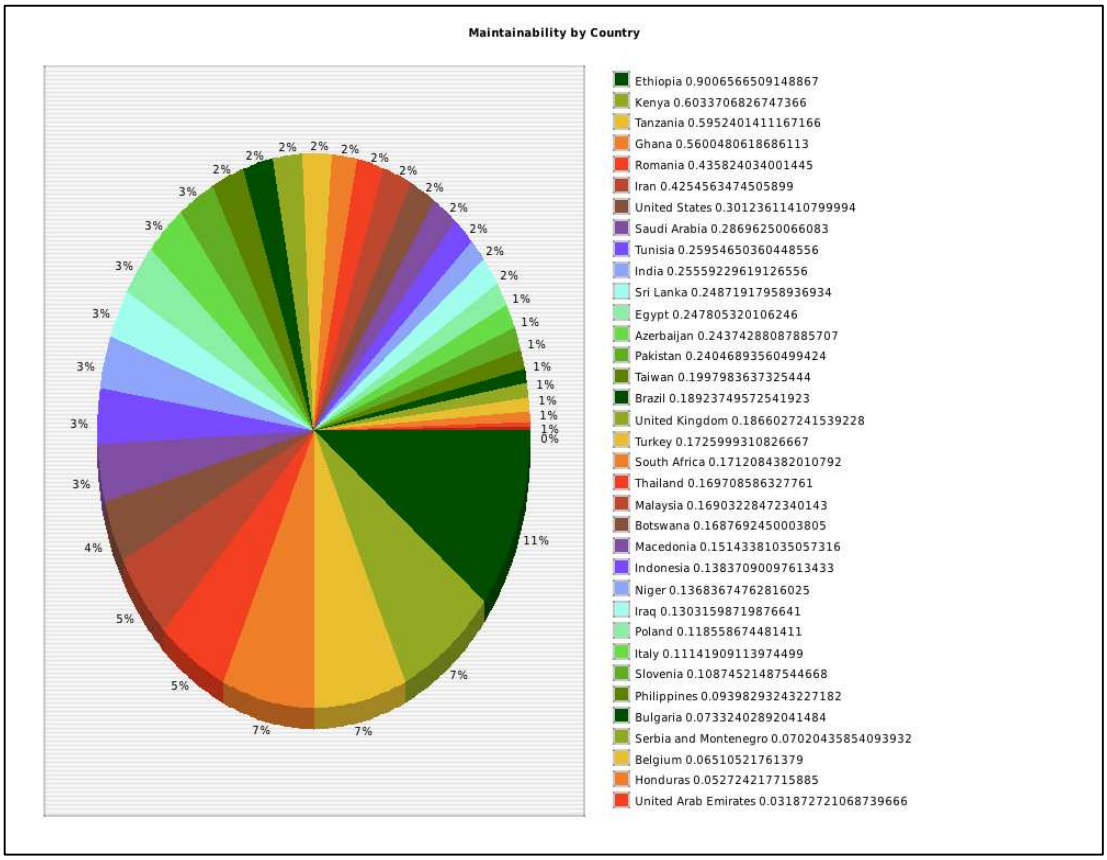

Figure 4.12 Maintainability of programs by countries

4.3.6 Understandability

Understandability measures how program be easy understand. This factor applies to all programs source code. This quality factor also programmer oriented, not user oriented. This quality factor is not confused with modularity. Because Program must be understand firstly, then it may be modified.

Many project manager and tester use the Halstead's effort for determination of this quality factor. Yes may be it is true. But I think that number of comments line is first place that help programmer to understand the source code. If programmer knows the programming language is not fluently so he may understand source code and may find the place of module, function and variables with the help of comments. So in this thesis we use division of CLOC to SLOC for determination this quality factor. Because more comments are not means that good programming. The understandability of programs by programming languages is as Figure 4.13. The understandability of programs by countries is as Figure 4.14

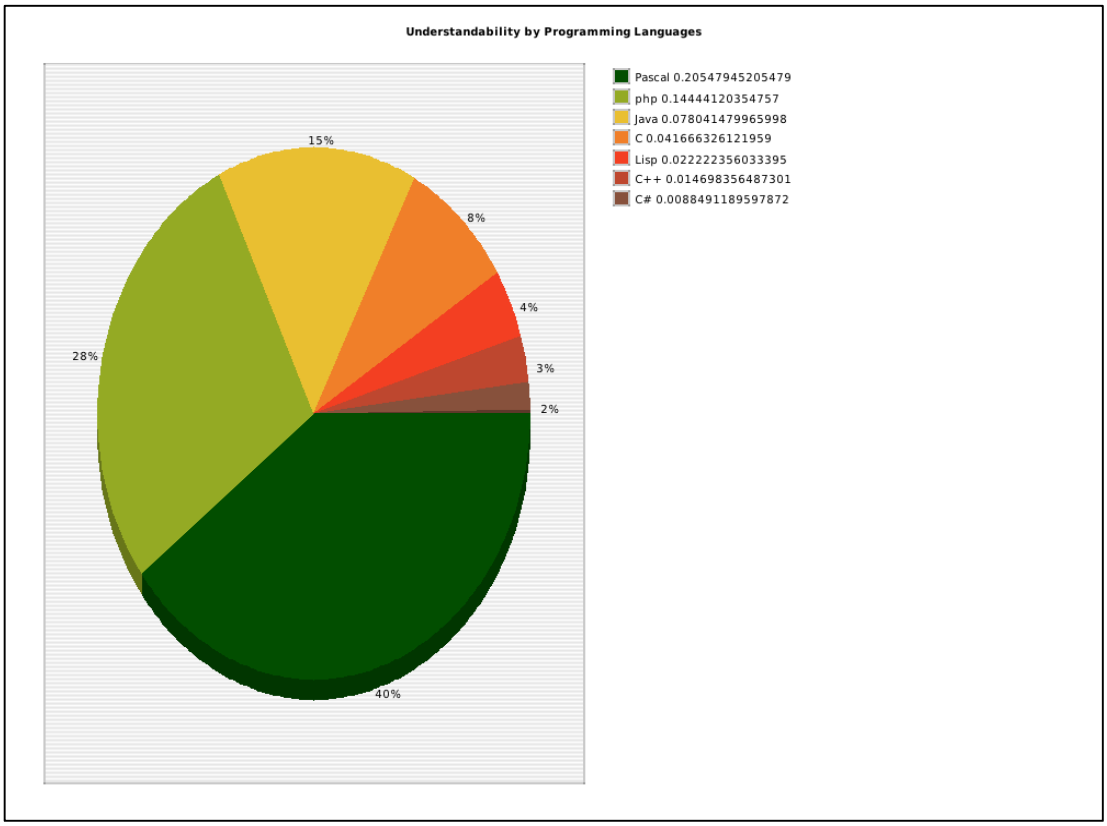

Figure 4.13 Understandability of programs by programming languages

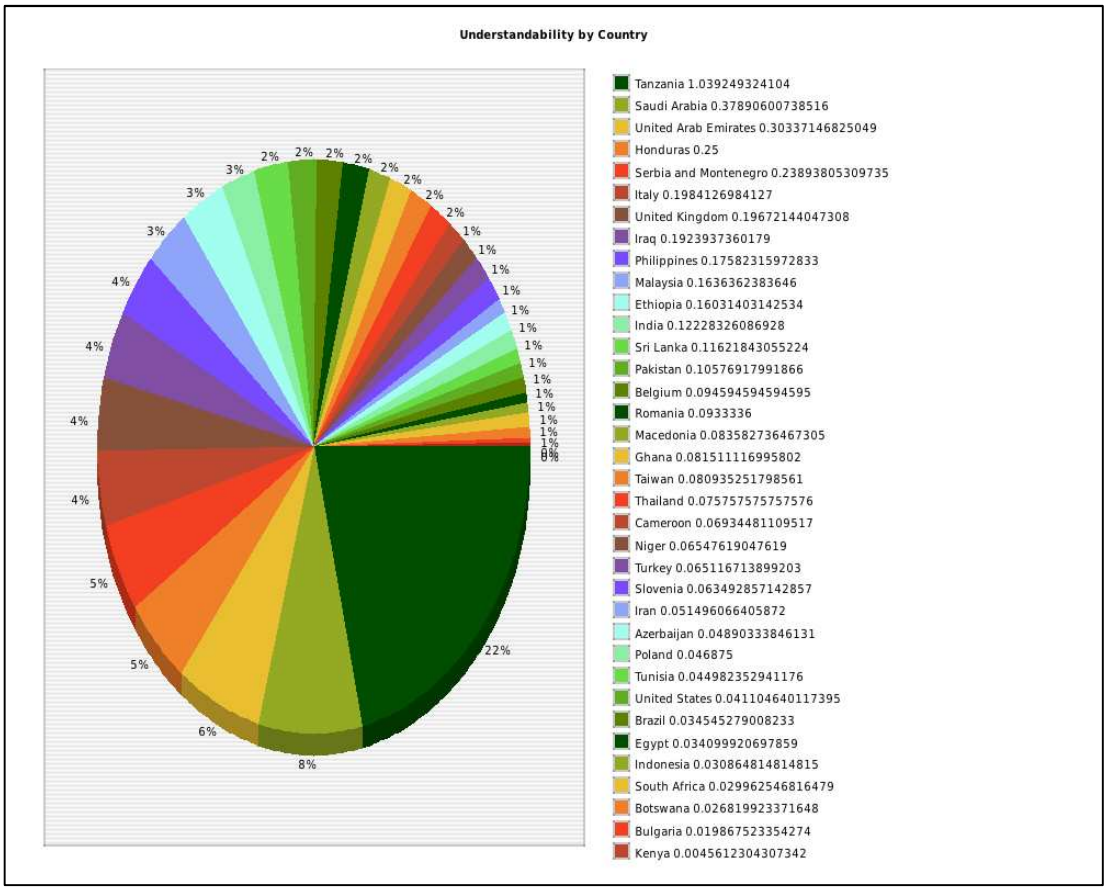

Figure 4.14 Understandability of programs by countries.

4.3.7 Performance of Programmers

Performance quality factor is divided into two parts. Performance of program and performance of programmer. Performance of Program is based on utilization of resources. Program performance is concerned how the job is done, how program is working and etc. This quality factor is determined by project manager and users. It is depended on number of bugs, design complexity, correctness and usability.

Performance of Programmers measures how programmer writes this program. Many project manager use the effort for determination of this quality factor. But it gives us only clarity, accuracy of program. The programmer may understand code clearly but he spends many time for coding. So time is important for determination of this quality factor. Halstead's time metric relates with this quality factor. It measures the time that programmer expends for writing program. So in thesis we calculate performance the division of time to SLOC. The performance of programmers by

programming languages is as Figure 4.15. The performance of programmers by countries is as Figure 4.16

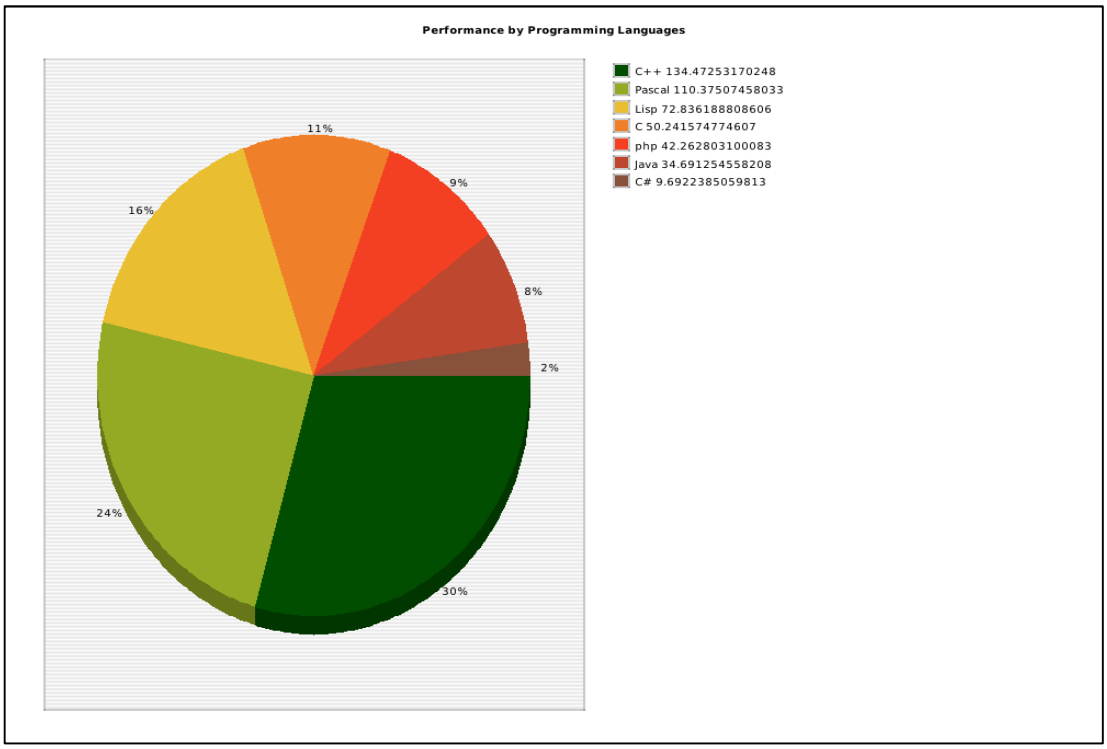

Figure 3.15 Performance of programmers by programming languages

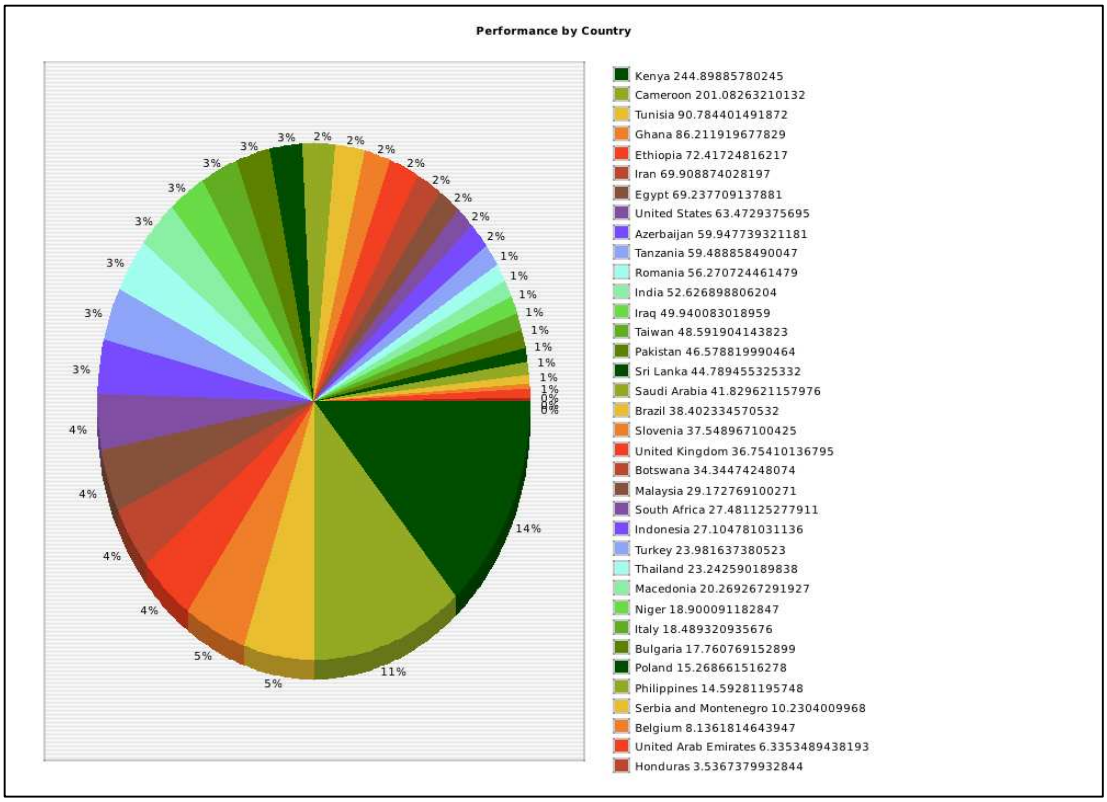

Figure 4.16 Performance of programmers by countries

4.3.8 Ability of Programmers

This quality factor measures the programmer ability which how he understand the program before coding. Firstly programmer must understand the program. After understanding it must create algorithm of problem. Then must choose the programming language which is be easy for writing the code. And at last programmer start to code the program. This quality factor applies all types of source code.

The determination of quality factor is used Halstead's program level. This metric also measures how programmer understands the problem before coding. This quality factor concerned with clarity and accuracy of program. But this type of quality factor is based on programmer not program. The ability of programmers by programming languages is as Figure 4.17. The ability of programmers by countries is as Figure 4.18

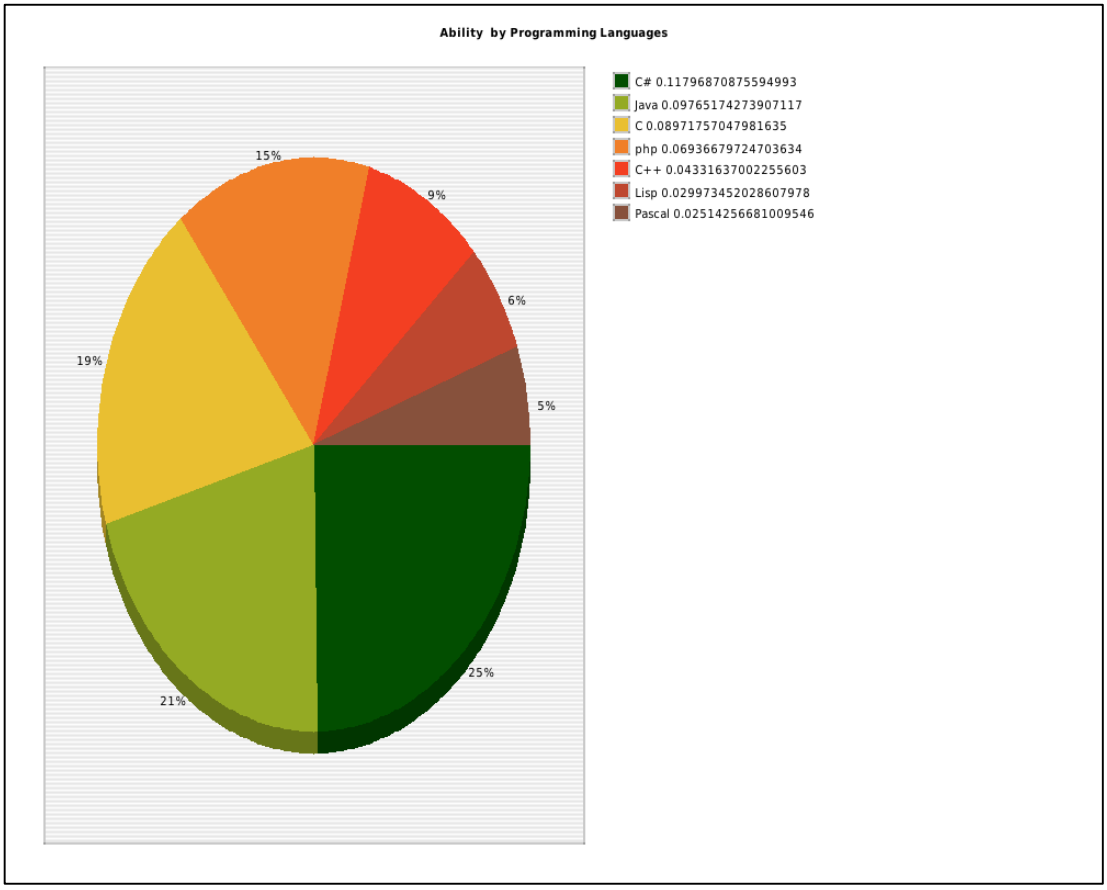

Figure 4.17 Ability of programmers by programming languages.

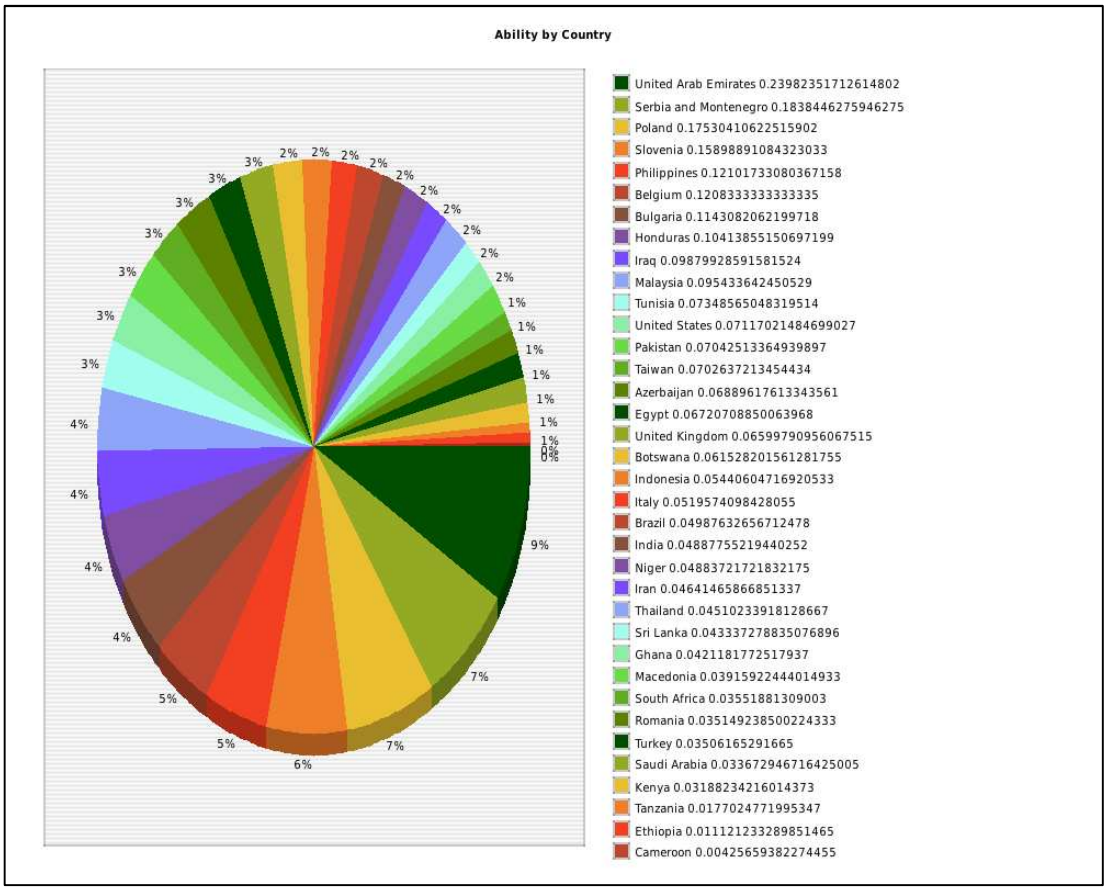

Figure 4.18 Ability of programmers by countries

4.3.9 Reliability of Programmers

Mainly tester measures the reliability of program. And reliability of program is not concerned with the accuracy and correctness. Reliability of programmers concerned to the function of program. This quality factor measure the program that how it works without a failure for at given period of time.

Reliability of Programmers is measured the ability of programmers that how many failure makes through the programming time. Some tester does not determine this quality factor. High number of bugs per programming time means that programmer is unreliable. Division of Halstead's bug number to programming time is determining reliability of programmers. The unreliability of programmers by programming languages is as Figure 4.19. The unreliability of programmers by countries is as Figure 4.20

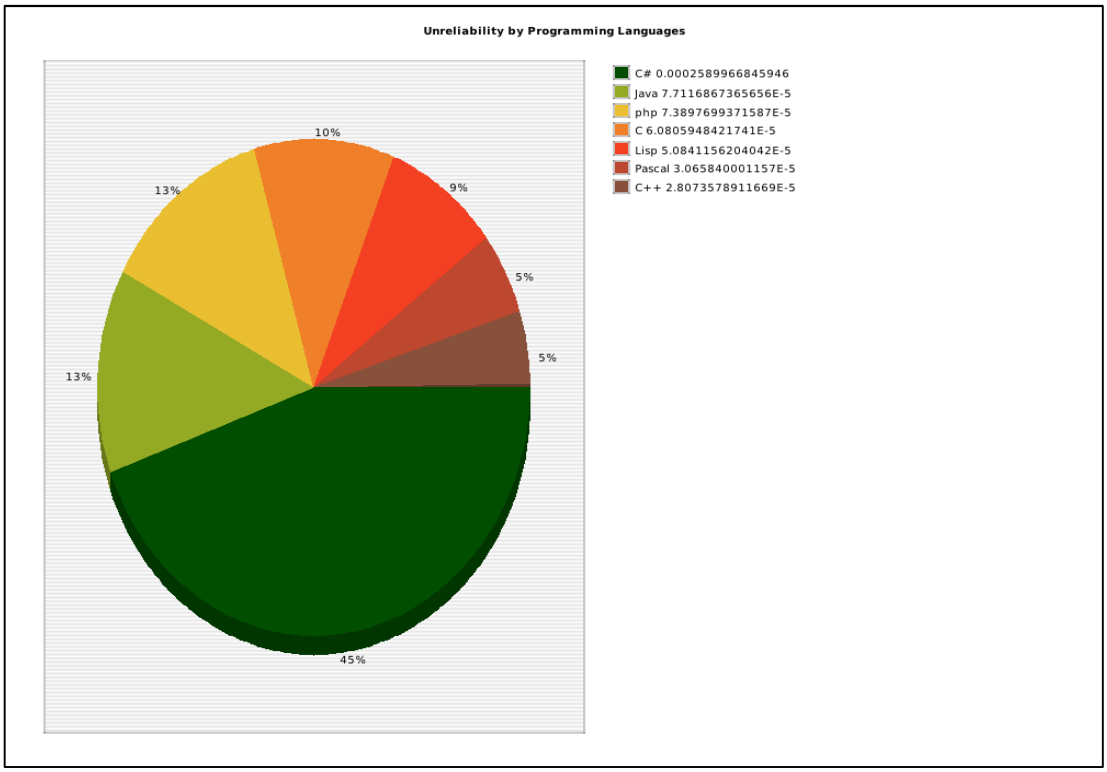

Figure 4.19 Unreliability of programmers by programming languages

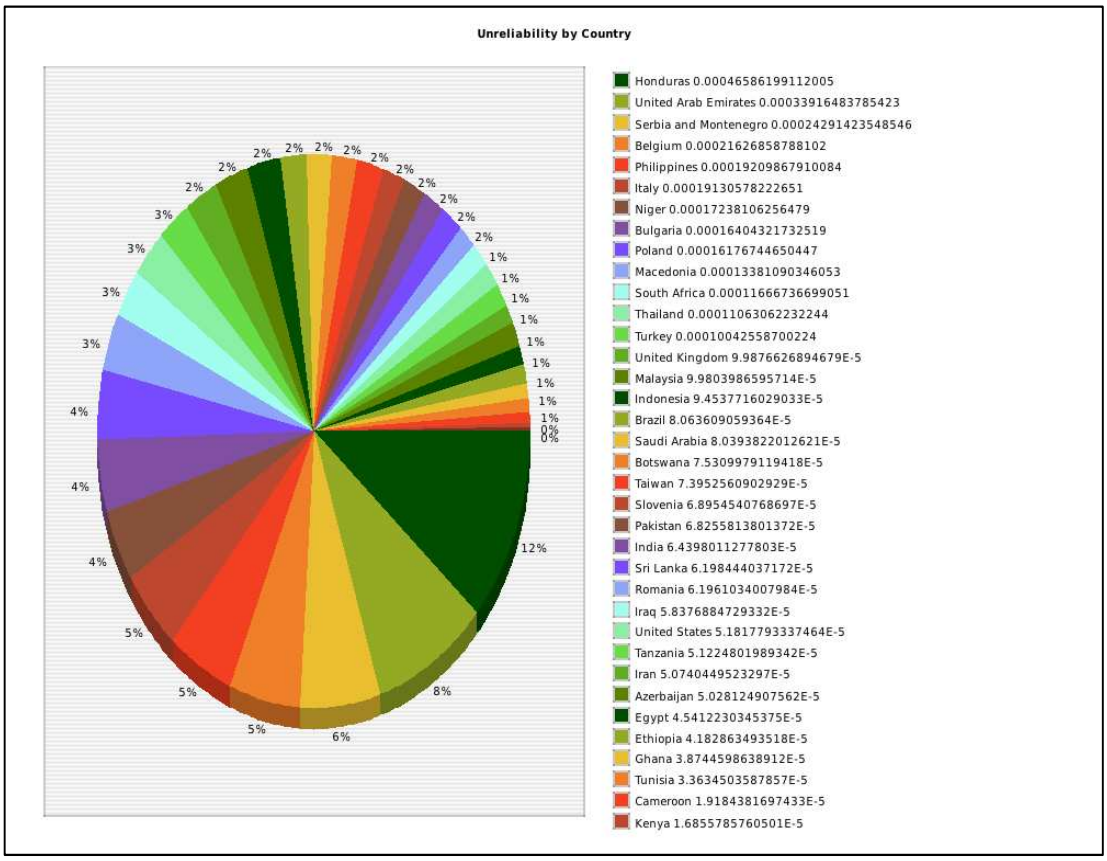

Figure 4.20 Unreliability of programmers by countries.

4.3.10 Accuracy of Programmer

Accuracy is a quality factor that has two types: Accuracy of Programmer and accuracy of program. Accuracy of program measures the precision of output. Accuracy is important between quality factors because the result of output is more significance for user. This quality factor applies all types of source code.

Accuracy of programmer determines how programmer seriously writes code. This quality factor affects to the complexity, modularity and understandability. The accuracy of programmer is determined by LOC metrics. Division Blank line of code to Source line of code defines the value of accuracy. This factor is also define as inaccuracy or unserious. The inaccuracy of programmers by programming languages is as Figure 4.21. The inaccuracy of programmers by countries is as Figure 4.22

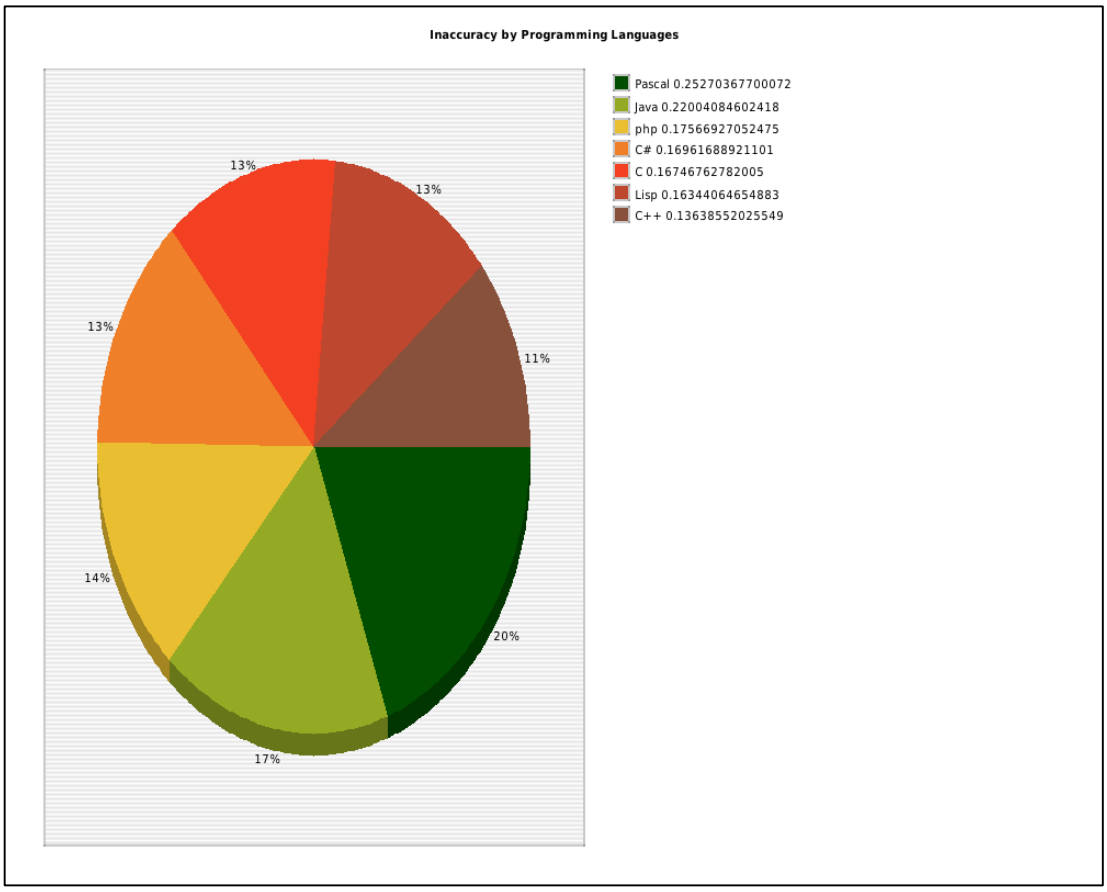

Figure 4.21 Inaccuracy of programmers by programming languages

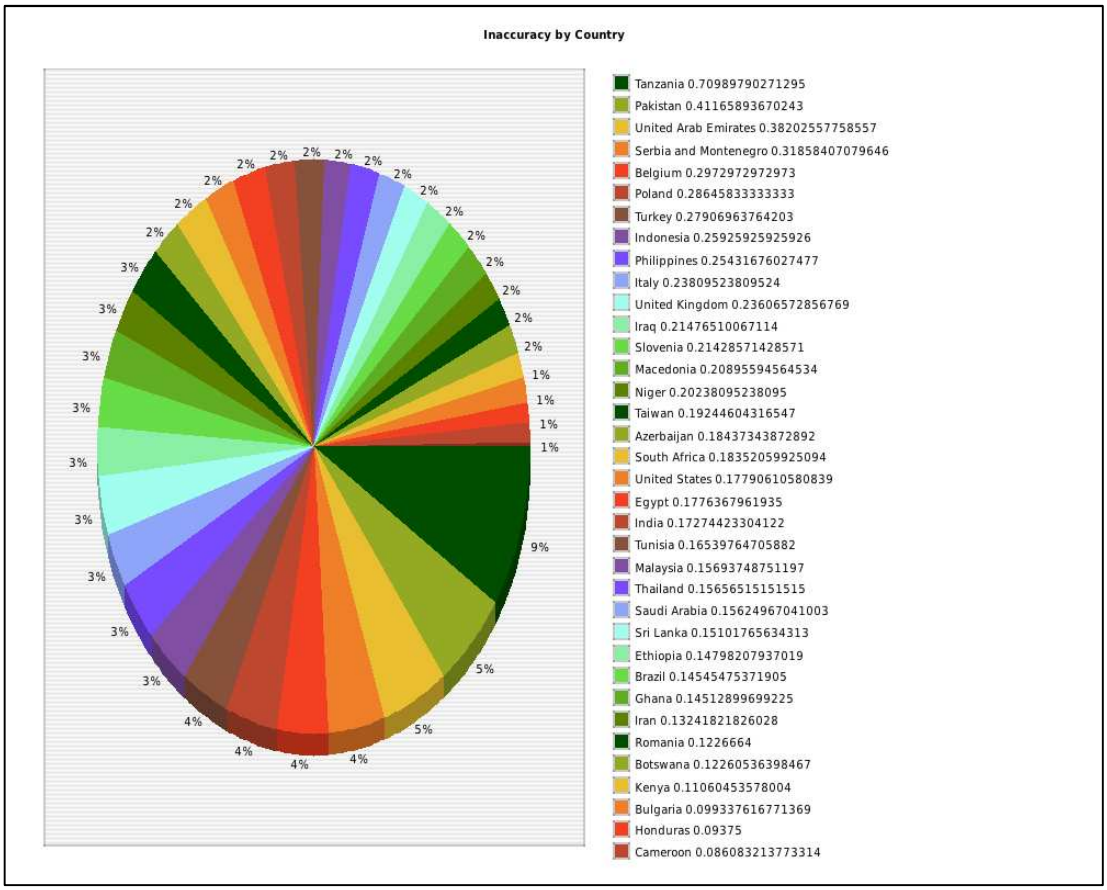

Figure 4.22 Inaccuracy of programmers by countries

4.3.11 Productivity of Programmer

Productivity is also divided into two parts. They are Productivity of program and programmers. Productivity of program measures the number of program's code line. If the number of source line of code is higher then the program is more productive. But in modern programming Productivity is not determined as this rule. If program is effectiveness then this is also productive.

Productivity of Programmers determines how many source code lines programmer writes during programming time. Division source line of code to Halstead's programming time measures productivity of programmers. This quality factor applies all types of source code. The productivity of programmers by programming languages is as Figure 4.23. The productivity of programmers by countries is as Figure 4.24

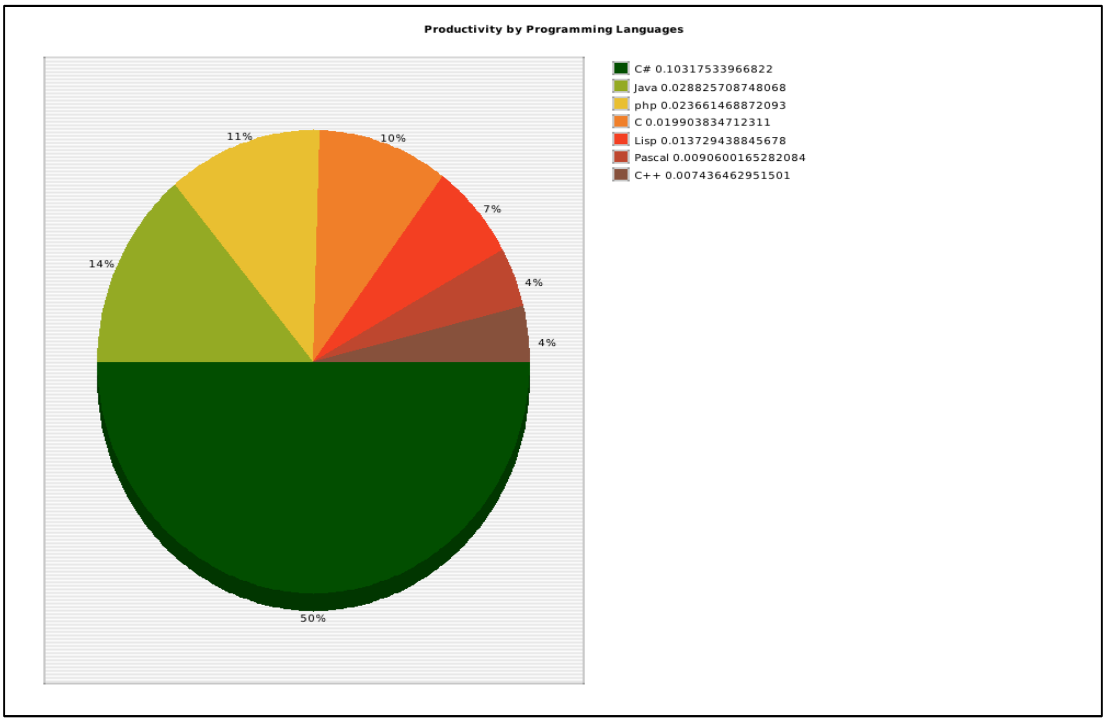

Figure 4.23 Productivity of programmers by programming languages

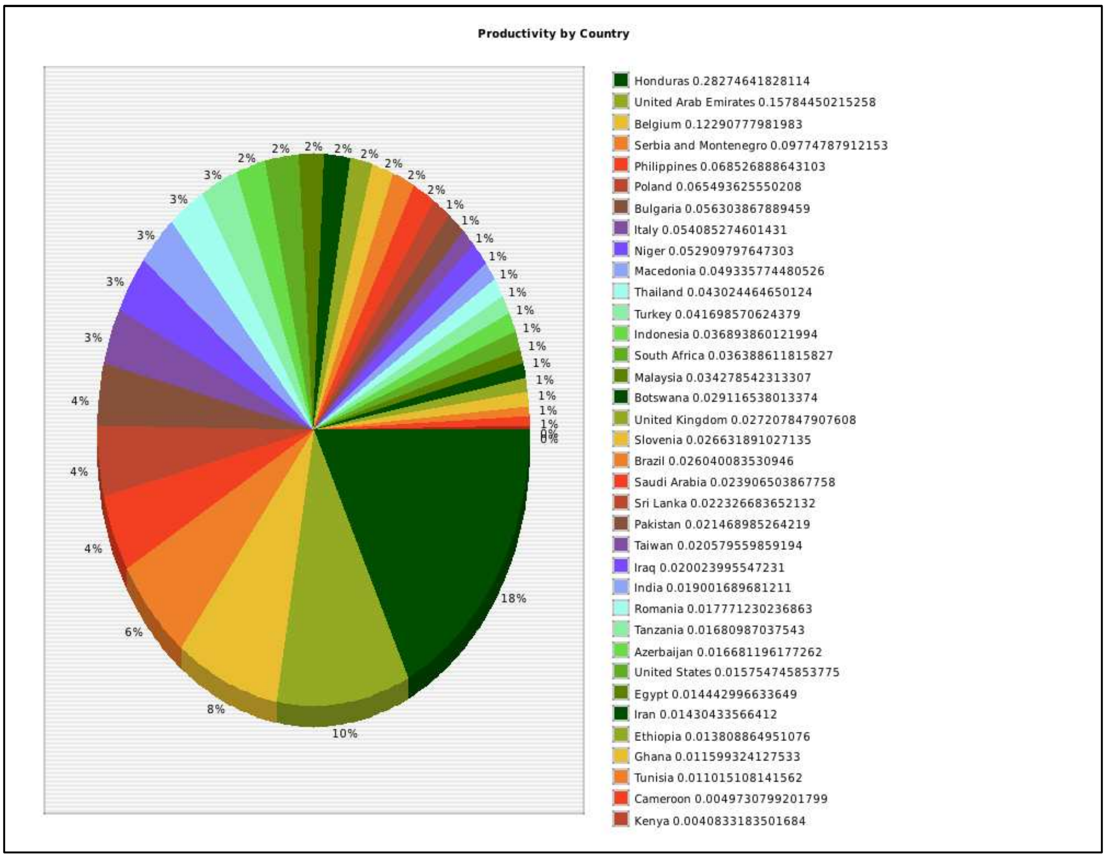

Figure 4.24 Productivity of programmers by countries

CHAPTER FIVE

CONCLUSION

The web tool calculates the modern software metrics and quality factors. The Halstead's metric and LOC are used in this calculation. The Halstead's metric is used rarely, notwithstanding LOC is used in modern analysis. Some of tester company thinks that it is impossible using older analysis metrics in modern programming. Because the programming style changes year after year. Yes in this idea they are right. But modern programming also is used operators, operands and etc. All new programming type is based on old one. In old programming all operations did some operation as manipulation, reading, writing and some mathematical and technical operations. Differentness of modern programming is these operations are done by class and functions. And these classes every time are saved on language's library. So these classes may be counted place of operations. But if programmers write own class then we must count inside operations.

In this thesis the quality factors are determined by programming languages and countries. The program collects 318 data from 95 programmers. The analyze tool works on online. So tester cannot see the source code of programs. At the results of this calculation it is seen that without seeing code it is possible determine the quality factor of programs. Only product qualities are determined in this research. Because the data for determination other quality factors, it must be collected from user.

REFERENCES

Alqutaish, R.E & Abran, A. (2005). An analysis of the designs and the definitions of the Halstead's metrics. In *Proceedings of the 15th International Workshop on Software Measurement (IWSM'2005) Montreal, Canada. pp. 337- 3527.*

Basell, V.R & Selby, R.W. (1985). Calculation and use of an environment's characteristic software metric set. *IEEE.*

Capers, J. (2012). *A short history of the Lines of Code (LOC) metric*. Namcook Analytics LLC. Retrieved April 8, 2013 from http://www.ifpug.org/ Documents/ Jones-LinesofCodeMetricV6.pdf.

Everald, E.M. (1988). *Software metrics*. Software Engineering Institute, Carnegie Mellon University.

Falcone, G. (2010). Software measurement. In *Hierarchy-Aware Software Metrics in Component Composition Hierarchies* (97-140). Berlin: Logos Verlag.

Hendriks, R., Vonderen, V.R. & Veenendaal, V. E. (2000). Measuring software product quality during testing. *Conference Proceedings European Software Quality Week.*

Kamaljit, K., Kirti, M., Neha, M. & Namita, K. (2009). Static and dynamic complexity analysis of software metrics. *World Academy of Science, Engineering and Technology 56*

Kaner, C. & Bond, W. P. (2004). Software engineering metrics: What do they measure and how do we know? *10th Internatıonal Software Metrıcs Symposıum.*

Kaushal, B., Vinit, T. & Pushpraj, P. (2012). Analysis of source line code (SLOC) metric. *International Journal of Emerging Technology and Advanced Engineering. ISSN 2250-2459, Volume 2,* Issue 5.

Park, R.E. (1992). The Size Subgroup of the Software Metrics Definition Working group & the Software Process Measurement Project Team. *Software size measurement: A framework for counting source statements.* Software Engineering Institute Carnegie Mellon University

Singh, G., Singh, D. & Singh, V. (2011). A study of software metrics. *IJCEM International Journal of Computational Engineering & Management*, Vol. 11.

Software Quality - Definitions. (n.d.). Retrieved May, 2013, from http://en.wikipedia.org/wiki/Software_quality#Definitions

Stephen, H. K. (2002). Software quality metrics overview. In *Metrics and Models in Software Quality Engineering* (2nd edition. Chapter 4). Addison - Wesley Professional

Tayyaba, N. (2011). Impact of user satisfaction on software quality in use. *International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11* No: 03

U.S. Department of Transportation Federal Aviation Administration. (1991). *Software quality metrics***.** Final Report