

# **Analytical Based Model to Measure Software Engineer's Productivity**

Gobikrishnan Thavarajah

(169138G)

Degree of Master of Business Administration in Information Technology

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 2018

# **Analytical Based Model to Measure Software Engineer's Productivity**

Gobikrishnan Thavarajah

(169138G)

The dissertation was submitted to the Department of Computer Science and Engineering of the University of Moratuwa in partial fulfilment of the requirement for the Degree of Master of Business Administration in Information Technology.

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 2018

## DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

.....

Date:.....

Gobikrishnan Thavarajah

The above candidate has carried out research for the Master's thesis under my supervision.

.....

Date .....

Dr. Amal Shehan Perera

## **COPYRIGHT STATEMENT**

I hereby grant the University of Moratuwa the right to archive and to make available my thesis or dissertation in whole or part in the University Libraries in all forms of media, subject to the provisions of the current copyright act of Sri Lanka. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

.....

Gobikrishnan Thavarajah

## **ABSTRACT**

In the software engineering industry one of the most central business factors is software developer's productivity, the understanding of the term productivity in the context of software development is not clearly defined, however, which cannot be measured cannot be managed, hence, software engineering companies from startup to enterprise are trying their level best to measure software developer's productivity level.

In order to solve this issue, everyone should have an understanding about software engineer's productivity, and also common as well as important factors which could act as an indicator to software developer productivity should be identified and validated. Considering the nature of the problem, a single factor cannot be considered as an indicator of a developer's productivity. Hence a multifactor model should be identified, validated and fine-tuned to produce better accuracy.

As part of this research, a survey among software developers was conducted in order to build a multifactor model which can be used to measure developer's productivity; the model was validated with real software development data and calibrate to producer more accurate result.

## **ACKNOWLEDGEMENT**

It is with immense gratitude that I acknowledge the guidance of my Supervisor, Dr. Shehan Perera. I am indebted to him for his advice, helpful comments and for the suggestions offered to me throughout the research study.

I am especially grateful to Dr. Chandana Gamage and Ms. Jeeva Padmini for sparing valuable time and providing continuous support for the identification of the research problem, formulating the conceptual model, survey data analysis, and reviewing research progress which contributed a lot to carry out the research successfully.

My sincere gratitude is also extended to all the academic and non-academic staff of Department of Computer Science & Engineering, The University of Moratuwa and my colleagues from the MBA in IT degree program for their support in numerous ways.

My thanks are also due to all my friends in 99X Technology, Tiqri Corporation Pvt Ltd, Millennium IT, who helped me in the most important and difficult task of gathering data during the limited period. I would also like to thank all those who responded to my questionnaire.

Gobikrishnan Thavarajah

(169138G)

# TABLE OF CONTENTS

DECLARATION.....	I
COPYRIGHT STATEMENT .....	II
ABSTRACT.....	III
ACKNOWLEDGEMENT.....	IV
TABLE OF CONTENTS .....	V
LIST OF FIGURES .....	IX
LIST OF TABLES .....	X
LIST OF ABBREVIATIONS .....	XI
1. INTRODUCTION .....	1
1.1. Background .....	1
1.1.1. Motivation .....	3
1.1.2. Research Scope .....	3
1.2. Problem Statement .....	4
1.2.1. Research Objectives .....	5
1.2.2. Research significance.....	5
1.2.3. Outline.....	6
2. LITERATURE REVIEW .....	7
2.1 The Definition of Productivity .....	7
2.1.1. The productivity of a developer .....	9
2.2. Factors Indicating the Quality and Quantity .....	10
2.2.1. Lines of code .....	11
2.2.2. Code quality .....	11
2.2.3. Code complexity .....	12
2.2.4. Function points.....	13
2.2.5. Defects.....	13
2.2.6. Effort estimation.....	14
2.3. Challenges in Measuring the Productivity .....	15
2.3.1. Performance measurements of value creation are missing .....	16
2.3.2. No productivity measurements on an r&d level are found .....	16
2.3.3. Performance measurement process is missing.....	17
2.4. Agile Software Development Methodology.....	17

2.4.1.	Scrum framework.....	18
2.4.2.	Existing performance metric and KPIs .....	20
2.5.	Summary .....	22
3.	RESEARCH METHODOLOGY.....	23
3.1.	Research Problem.....	23
3.1.1.	Research method .....	24
3.1.2.	Conceptual framework of the research .....	26
3.2.	Development of Hypotheses.....	28
3.3.	Operationalization .....	28
3.3.1.	Population and sample selection .....	30
3.3.2.	The process of data collection.....	32
3.4.	Summary .....	33
4.	DATA ANALYSIS.....	34
4.1.	Introduction .....	34
4.2.	Data Gathered from Software Development Activity.....	34
4.3.	Descriptive Statistics for Development Activity Data .....	35
4.3.1.	Development activity data by age .....	35
4.3.2.	Sample of software engineers categorized by experience level.....	36
4.3.3.	Sample date of software engineers categorized by gender .....	37
4.4.	Extraction and transformation of development activity data .....	37
4.4.1.	Code quantity .....	37
4.4.2.	Code quality .....	38
4.4.3.	Code complexity .....	40
4.4.4.	Work effort.....	41
4.4.5.	Productivity.....	41
4.5.	Testing Hypothesis - Pearson's Correlation Analysis.....	41
4.5.1.	The correlation between code quality and productivity.....	42
4.5.2.	The correlation between code quantity and productivity.....	43
4.5.3.	The correlation between code complexity and productivity .....	43
4.5.4.	The correlation between actual hours worked and productivity of a software developer.....	44
4.5.5.	The logarithm value of correlation between code quality and productivity.....	45



4.5.6.	The logarithm value of correlation between code quantity and productivity.....	46
4.5.7.	The logarithm value of correlation between code complexity and productivity.....	47
4.5.8.	The logarithm value of correlation between actual hours worked and productivity of a software developer .....	48
4.6.	Linear Regression Analysis.....	49
4.7.	Reliability of Survey Data.....	50
4.8.	Descriptive Statistics for Survey Demographic Data.....	51
4.8.1.	Sample of software engineers grouped by age.....	51
4.8.2.	Sample of software engineers categorized by gender .....	52
4.8.3.	Sample of software engineers categorized by experience level.....	53
4.9.	Presentation of Variable Related Sections Information .....	53
4.9.1.	Quality and software productivity.....	53
4.9.2.	Quantity and software productivity.....	54
4.9.3.	Code complexity for software productivity .....	55
4.9.4.	Work effort for software productivity.....	56
4.10.	Testing Hypothesis - Pearson’s Correlation Analysis .....	57
4.10.1.	The correlation between code quality and productivity of a software developer	58
4.10.2.	The correlation between code quantity and productivity of a software developer	59
4.10.3.	The correlation between code complexity and productivity of a software developer.....	60
4.10.4.	The correlation between minimal work effort and productivity of a software developer.....	61
4.11.	Summary.....	61
5.	RECOMMENDATIONS AND CONCLUSION.....	63
5.1.	Introduction .....	63
5.1.1.	Research conclusion one .....	63
5.1.2.	Research conclusion two.....	63
5.1.3.	Research conclusion three.....	64
5.1.4.	Research conclusion four .....	64
5.2.	Research Assumptions and Limitations .....	64
5.3.	Recommendation.....	65

5.4. Suggestion for Further Research .....	65
REFERENCES .....	67
APPENDIX A: TITLE.....	71
APPENDIX B: TITLE.....	76

## LIST OF FIGURES

Figure 1 Preliminary steps of Research .....	8
Figure 2 Research method.....	25
Figure 3 Overall IT workforce by job category .....	31
Figure 4 Process of data collection .....	33
Figure 5 – Software development team sample grouped by age .....	35
Figure 6 - A sample of software engineers categorised by experience level .....	36
Figure 7 - A Sample Date of Software Engineers Categorized by Gender.....	37
Figure 8 - The correlation between Code quality and productivity .....	42
Figure 9 - The correlation between Code quantity and productivity .....	43
Figure 10 - Correlation between code complexity and productivity.....	44
Figure 11 -Correlation between minimal work effort and productivity.....	44
Figure 12 -The correlation between Code quality and productivity .....	45
Figure 13 - The correlation between Code quantity and productivity .....	46
Figure 14 - Correlation between code complexity and productivity.....	47
Figure 15 - Correlation between minimal work effort and productivity.....	48
Figure 16 - Non-Liner Regression Analysis .....	49
Figure 17 Model Summary .....	50
Figure 18 Sample of software engineers grouped by age .....	51
Figure 19 Sample of software engineers categorized by gender.....	52
Figure 20 Sample of software engineers categorized by experience level .....	53
Figure 21 Quality and software productivity .....	54
Figure 22 Quantity and software productivity .....	55
Figure 23 Code complexity for software productivity.....	56
Figure 24 Work effort for software productivity .....	57
Figure 25 Survey Questions part 1.....	71
Figure 26 Survey Questions part 2.....	72
Figure 27 Survey Questions part 3.....	73
Figure 28 Survey Questions part 4.....	74
Figure 29 Survey Questions part 5.....	75

## LIST OF TABLES

Table 1 Factors in the conceptual framework .....	26
Table 2 Research hypotheses .....	28
Table 3 Operationalization.....	28
Table 4 Reliability of surveys data.....	50
Table 5 The correlation between Code quality and productivity.....	58
Table 6 The correlation between Code quantity and productivity.....	59
Table 7 Correlation between code complexity and productivity .....	60
Table 8 Correlation between minimal work effort and productivity .....	61
Table 9 Correlation Values .....	63

## LIST OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Description</b>
ICT	: Information and Communication Technology
LOC	: Lines of Code
IT	: Information Technology
SLOC	: Source Lines of Code
CNN	: Cyclometric Complexity Number
R&D	: Research and development
MTBF	: Mean Time Between Failures
CAC	: Cronbach's Alpha Coefficient
SQA	: Software Quality Assurance

# 1. INTRODUCTION

## 1.1. Background

Productivity measurements related to the efficiency and effectiveness of an individual or a team has received a lot of research attention and are generally considered to be an important part of any high performing organisation. Although many organisations are successful in developing, selling, and delivering products, we also observe that a substantial part of the software product development projects fails. Failure can be in delivering late, or with insufficient quality, or not delivering at all. To improve the success rate of software product development projects, the connection between success/failure and the performance of the organisation needs to be understood and used for decisions. (Goparaju Purna Sudhakara, Ayesha Farooq .b and Sanghamitra Patnaik .c, 2012)

In a volatile marketplace, the organisations should be prepared to handle and respond to the changeable and complex customer requirements, personnel, cost, and schedule. Constant schedule pressure, simultaneous work in many projects, chasing deadlines, customers changing requirements, and demand for new skills and knowledge, continuous code inspections, and sudden offshore assignments keep the developers under continuous stress. In the meantime, they are expected to be proactive, flexible, adaptable, share knowledge, and follow professional practices. Despite undergoing stress and increasing expectation, developers having the inner aptitude and behavioural traits can increase their performance. Researchers assert that developer's performance and project successes depend on their commitment, initiative, leadership, personality, and intrinsic motivation. (Chris Peck, Dale W. Callahan, 2002)

Over the period, through the evolution of software development process Agile methodologies been introduced. The flexibility provided by incorporating agile software development approaches in software development processes. Operating agile means should be able to rapidly and inherently create, respond and embrace change in business as well as technical context, Agile approaches encourage the developers to

learn from experience and add to customer values by reducing cost, improving quality and maintaining simplicity while deemphasising long-term planning in favour of short-term adaptiveness.

Management layer is always keen to know and monitor the productivity from the micro level in their organisations. The more productive the employees, more work can be completed in a short period which brings more money and high customer success. Maximizing the team's productivity is one of the highest responsibility of a scrum master or project manager. In fact, there is a common phrase, "you cannot plan if you cannot measure". Usually, software development organisations evaluate the productivity of a software developer considering their contribution at different levels.

Few reasons why organisations tend to measure software developer's productivity at an organisational level? (Inga Podjavo, Solvita Berzisa,2017)

- Assess competitiveness with other organisations
- Track and evaluate progress over time
- Support performance evaluation of software executives
- Support bonus allocation among software executives
- Decide allocation of resources to onshore/offshore/outsourced

Few reasons why organisations tend to measure software developer's productivity at the team level? (Inga Podjavo, Solvita Berzisa,2017)

- Compare teams to identify performance gaps
- Provide support performance evaluations
- To decide the bonus allocations

Few reasons why organisation tend to measure software developer's productivity at the individual level? (Inga Podjavo, Solvita Berzisa,2017)

- Support allocation of resources across the teams
- Contribute to individual performance review process
- Support allocation of bonuses among individual contributors

### **1.1.1. Motivation**

In the current computer-driven world, there are many software engineering companies producing software products as well as providing services to other organisation to support digitalisation. The competition among the software engineering organisation is usually higher compared to other industries. To face this competition every organisation around the world seeking all the possibilities to improve them self and position themselves on the top of the completion.

To become the best of the best, it is necessary to measure the current strength and weakness. The backbone of each software engineering organisation is the organisation's employees, especially the software engineering professionals who produce the code which software will function on. Since it is necessary to measure and keep track of the effectiveness and productivity of software engineers, based on the different organisation's culture and nature, evaluating and measuring the productivity of software engineers differs. (Chris Peck, Dale W. Callahan, 2002)

The motivation of this research is to find out the factors which can be used as an indicator to measure the productivity of a software developer, identify most important factors and create a standard model which can be used in an agile scrum environment to measure the software developer's productivity.

### **1.1.2. Research Scope**

The scope of this research is to find out

- What productivity means for software engineering professionals.
- Identify some of the effective matrices available today to measure productivity in an agile environment.
- Propose a suitable model to measure software engineer's productivity in an agile environment.



## 1.2. Problem Statement

Today's globalisation world many software engineering companies, especially independent software vendors are more interested to have distributed development teams since keep tracking of productivity becomes a vital part for the management. By measuring the productivity, the management is trying to achieve the following advantages. (Goparaju Purna Sudhakara, Ayesha Farooq .b and Sanghamitra Patnaik .c, 2012)

- Reduce the software development cost.
- Diversified (international) experience and expertise.
- More efficient workflow.
- Hire the best talent.

When it comes to the distributed team, monitoring the team and managing the project, needs the correct approach. The most crucial question which project sponsors have is; how to measure software developer's productivity? What are the common and important factors which can be used as an indicator to measure the productivity of a software developer?

The management team has to answer all of these questions to choose the right approach to manage software developers and improve their performance to gain the maximum output.

However, programming is not like other professions. We cannot measure it as we would measure some manufacturing process, where we could we count the number of correctly-made items rolling off the assembly line.

To measure the developers' productivity, it is vital to identify the factors which can be used to understand the quality and quantity of the software engineers' delivery.

### **1.2.1. Research Objectives**

- To explore what productivity means for software engineers working in scrum-based agile methodology.
- To explore factors which can be considered as an indicator for the productivity of software engineers.
- To explore the necessary actions which can be taken to improve the productivity based on the measurement outputs.

### **1.2.2. Research significance**

There are several well-known statements related to performance measurements in the literature. “What gets measured gets done” and “You are what you measure” are two classical examples of quotations related to the use of performance measurements. The paramount importance of evaluating the software developer’s productivity is generally acknowledged both in the literature and in practice.

The purpose of this research thesis is to fill the gap and remove the misconception when it comes to measuring the productivity of a software developer in the scrum-based agile environment by providing a common model which can be used to measure the software engineer’s productivity. (H.C. Shiva Prasad Damodar Suar, 2010)

The main focus of the performance measurement system is to provide managers with the needed information to be able to make conclusions about what actions to take to improve the performance of the organisation.

By identifying the productivity level of software engineers, the organization tends to achieve the following advantages.

- To identify the best suitable project development methodology which suits the team to produce a better result.
- To find the most cost-effective tools and techniques.
- To optimum the developer’s productivity by removing the impediments.
- To compare the internal team with industry competitors.

### **1.2.3. Outline**

Chapter one contains an introduction to the research study. It initially explores the background of productivity within the scope of software developers; this chapter also contains details of motivation, scope, objective and significance of this research.

Chapter two focuses on the existing literature regarding the approaches of measuring the productivity of software engineers, Qualitative and quantitative factors which can be used to study about software developer productivity. In this chapter details about Agile methodology and scrum framework also been discussed as a software development methodology.

Chapter three details about methodology using which the research has been done. This chapter also details about data which was gathered through the survey and interviews, details about population and sample selection, information about the process which was followed to collect the data.

Chapter four discusses insight created using the data gathered from the survey and code analysis. Furthermore, this chapter details the relationship between factors which can be used to measure the software developer's productivity.

Finally, Chapter five provides details about the conclusion which was taken as the result of this research thesis. Furthermore, this chapter elaborates on the recommendation about how organisations can improve the software developer's productivity.

## **2. LITERATURE REVIEW**

This chapter contains the literature survey which was used to identify the variables which could be an indicator of a software developer's productivity to develop the conceptual model at a later stage.

The Chapter is dedicated to capturing the literature available about productivity in the context of software developer, factors indicating the quality and quantity, challenges in measuring the productivity as well as the agile software development environment.

### **2.1 The Definition of Productivity**

First, it is important to introduce the main concept of productivity. The origins of the term "productivity" traced back to the eighteen century, and was introduced by Quesnay; however, until the middle of the past century, the definitions were blurred. Traditionally, productivity has been defined as the ratio of outputs produced per unit of input. This definition fits well in manufacturing paradigms hence it is based on quantities of standardised and identified units of measurement.

Productivity should be identified as a component of performance, not a synonym for it (Sink, Tuttle, & DeVries, 1984). This claim is argued from the concept of comparative productivity performance and not as a result unit; namely productivity measures should be used for comparison over time, while performance represents a timely measure. In this direction, the value of productivity measurement lies in the capability to manage and monitor, to reach a more efficient resources use. Also, as Nachum (1999) argued, the main objective of measuring productivity is to perform productivity enhancement. Moreover, productivity improvements should be reflected in ROI improvements. Therefore, productivity is inversely proportional to the costs incurred

As Anselmo and Ledgard (2003) pointed following Lord Kelvin's affirmation<sup>1</sup>, software productivity enhancement cannot be expected without productivity measurement. An appropriate productivity measure provides a tool as for how to

achieve productivity (Nachum, 1999). Furthermore, following Gummesson (1992) recommendation, before measuring productivity in the service industry, identification of what is to be captured is required. Thus, considering these contributions, to create a software engineering productivity measurement, distinguishment of factors, inputs and outputs susceptible to be measured is required.

The roadmap for this process may follow the flowchart of Figure 1. In this roadmap, there are some steps which could be carried out with quantitative research methodologies such as conducting the survey.

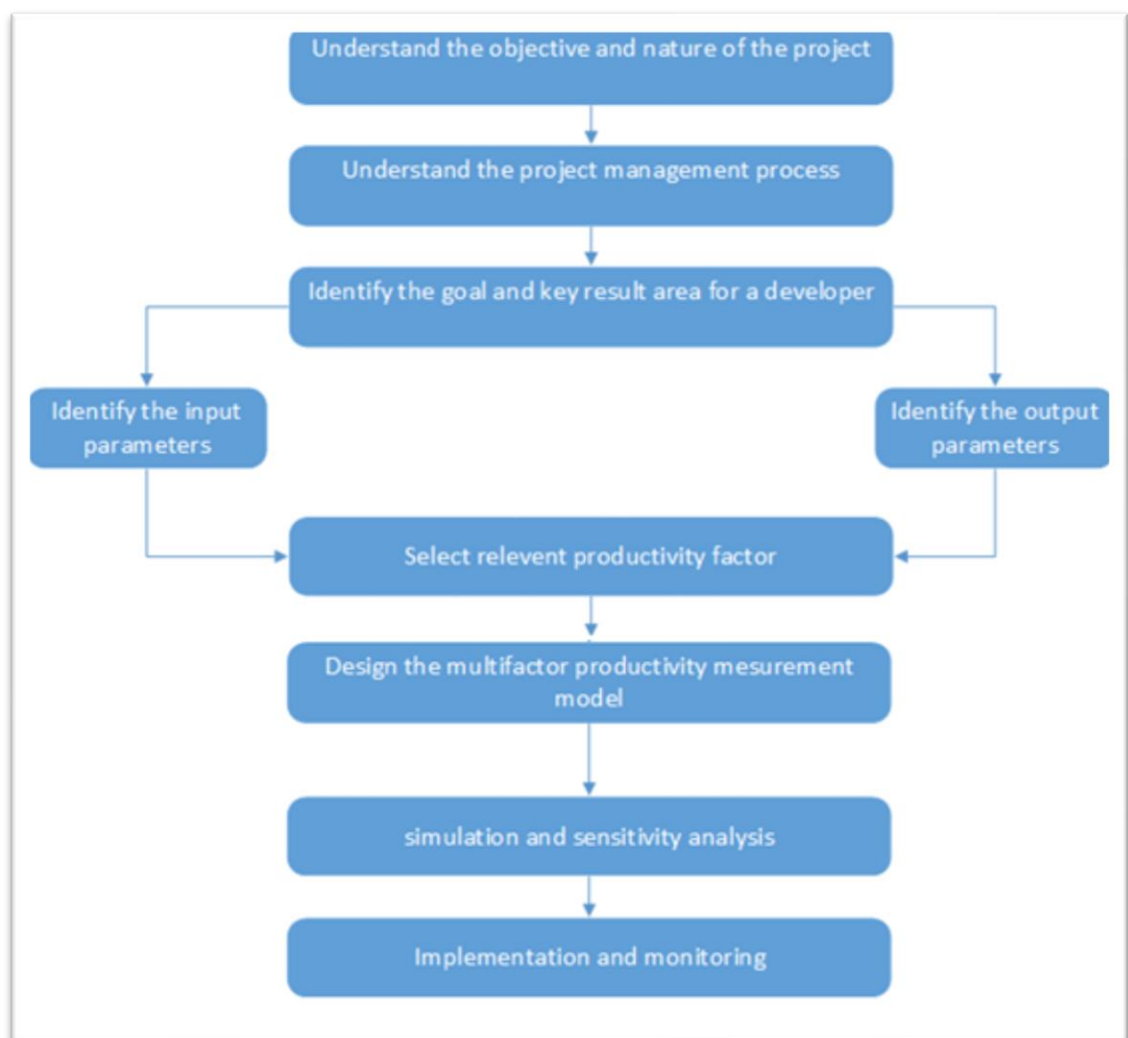


Figure 1 Preliminary steps of Research

### **2.1.1. The productivity of a developer**

"You cannot plan if you cannot measure." This is a concept still taught in business school; it is a mantra of many managers, It assumes everything a developer does is objectively and consistently measurable. As discussed above, there is no reliable, objective metric to measure developers' productivity.

It is obvious that some people are better than others. Better developers can be identified, but currently, there is no better number or rational ranking system available at the moment, objectively based on output, that consistently and reliably ranks developers.

The software development industry to a large extent is an open system where stakeholders, clients and the end users influence inputs and outputs, which produces a contribution to both the internal and external efficiency, and therefore the productivity measurement for software engineers needs a unique approach. (Machek Ondrej, Hnilica Jiri, and Hejda, 2012)

Inputs and outputs measurement should consider both quantity and quality aspects. This concept is reflected in the premises that Grönroos and Ojasalo established: "The better the perceived quality that is produced using a given amount of inputs (service provider's inputs and customers' inputs), the better the external efficiency is, resulting in improved service productivity" and "The more efficiently the service organization uses its resources as input into the processes and the better the organization can educate and guide customers to give process-supporting inputs to produce a given amount of output, the better the internal"

Most research in software engineering defines productivity along similar lines; here are some examples:

- number of modification requests and added lines of code per year,
- number of tasks per month,
- number of function points per month,
- number of source lines of code per developer hour,

- number of lines of code per person-month of coding effort,
- amount of tasks completed per reported hour of effort for each technology,
- the ratio of produced logical code lines and spent effort,
- average number of logical source statements output per month over the product development cycle,
- total equivalent lines of code per person-month,
- resolution time defined as the time, it took to resolve a particular modification request, and
- a number of editing events to a number of selection and navigation events needed to find where to edit code.

As Cambridge dictionary defines, productivity is the rate at which a company or country makes goods; this phrase can be translated as the following equation:

$$\text{Productivity} = \frac{\text{Output}}{\text{Input}}$$

Considering the above question as a base, and when considering the output of the survey, the following equation can be proposed as a common and basic model which can indicate a developer's productivity. Following factors are considered as an output:

- Quantity
- Quality
- Complexity

Actual hours worked is considered as a factor for the input category. (Chris Peck, Dale W. Callahan, 2002)

## **2.2. Factors Indicating the Quality and Quantity**

Measuring the software engineering productivity is a complex task. However, there are some straight forwards factors which can be used as an indicator to understand the work produced by the software developers.

As a result of many studies, it was proven that individually considering these factors to measure the productivity will not give a good and accurate result.

### **2.2.1. Lines of code**

Source lines of code, also known as lines of code, which is a software metric used to calculate the size of a computer program by considering the number of source lines in the software source code. Source line of code is typically used to predict the amount of work effort which requires to develop software, as well as to estimate programming productivity or maintainability once the software is produced.

Logical SLOC is trying to measure the number of executable "statements", but their specific definitions are tied to very specific programming languages (one simple logical SLOC measure for C-like programming languages is the number of statement-terminating semicolons). It is easier to create tools that can measure physical SLOC, and physical SLOC definitions are easier to explain. However, physical SLOC measures are sensitive to logically irrelevant formatting and style conventions, while logical SLOC is less sensitive to formatting and style conventions. However, SLOC measures are often stated without giving their definition, and logical SLOC can often be significantly different from physical SLOC. (Peck, C., & Callahan, D.) 2002.

### **2.2.2. Code quality**

A vital part of a good product is an efficient code; an inefficient code can make the end users frustrated, which will make a significant impact to the business, because of this risk there are many approaches to make sure the code which software engineers produce are efficient and follow the best practices. Some methods which are used to analyse the code quality are:

- Pear code Review
- Automated code quality measurement tools such as SonarQube

While doing the code quality check various aspect are verified to make sure the code is in good quality, some of the example such aspect includes

- Unit test coverage



- Duplication lines
- Architecture & Design
- Complexity
- Maintainability rating
- Reliability rating
- Potential Bugs
- Security rating

### **2.2.3. Code complexity**

Code complexity is a very important and vital factor which is the most commonly used unit of measurement calculated through cyclomatic complexity and commonly referred to known as cyclomatic complexity number or CNN. Traditionally, cyclomatic complexity is known as “McCabe”, since it was originally invented by the mastermind Tom McCabe in the year of 1976. The CNN is the number of all possible count of execution paths of a function written as the code. A function with only a single path which means a function without if statement or loops has one as a CNN count. CNN count increases if there many if statements, looping construction or any other decision points, deciding which code should be executed.

Mostly, its recommended to have CNN count below ten. Most of the tools available today in the market to measure the CNN count will not produce an accurate result if the CNN count goes above twenty. Functions which contains CNN value more than 20 are hard to test, and at the same time it is hard to maintain the code as well. Usually, functions which are hard to test and maintain tend to have more bugs.

An organization which test the code quality using tools checks every check-in for CNN values if the value is higher than the system will not allow the developer to merge the code into the repository (Peck, C., & Callahan, D.) 2002.

#### **2.2.4. Function points**

A function point is a measurement which expresses the business functionality of the system. Function points are considered to calculate a functional size measurement of software.

Function point analyze was introduced in the year of 1979 in “Measuring Application Development Productivity” by Allan Albrecht. While requirement elicitation, business needs are converted into functional requirements, then each functional requirement is categorized into five types. (Chris Peck, Dale W. Callahan, 2002)

1. Outputs
2. Inquiries
3. Input
4. Internal files
5. External interface

After identifying the category, then the task will be assessed for its complexity, and it will be assigned a number of function points. Each of this functional requirement maps to business needs, for an example of data input, data query etc.

#### **2.2.5. Defects**

After development is completed, the code will be a move to quality assurance. Quality assurance team will be testing the functionality using various testing methods. During the testing bugs which are identified will be listed with the appropriate severity types. (Scrum Alliance, 2016)

Bugs can be categorized into four severity levels:

- **Critical:** This defect indicates complete shut-down of the process, nothing can proceed further
- **Major:** It is a highly severe defect in the system. moreover, certain parts of the system remain functional
- **Medium:** It causes some undesirable behaviour, but the system is still functional

- Low: It will not cause any major break-down of the system

Bugs can be categorized into three priority type:

- Low: The Defect is an irritant, but repair can be done once the more serious Defect has been completed.
- Medium: defect can be fixed during the normal course of the development.
- High: The defect must be resolved urgently as it affects the system.

#### **2.2.6. Effort estimation**

Estimating the effort required and calculating the cost for that are a vital part of project management. The team cannot perform the planning if they do not do these estimations. As per today's dynamic software development trend, software developers tend to use external components such as already developed and freely available framework, modules, rather than building all the components from scratch. This trend has led to a new kind of estimation methods for development effort. Typically, estimation moved away from volume or size-based estimation to factional and component-based estimation.

In the industry, there are many estimation methods currently available, such as expert estimation: through this method, the estimation is made using expert judgmental process. Formal estimation model: this method is based on the mechanical process such as using a formula which is created using the historical data. Combination-based estimation: this method is based on the combination of both expert estimation as well as formal estimation model. (Scrum Alliance, 2016)

However, estimating the work required for a project or even single task is not an easy job to do, there are challenges in providing an accurate estimation, some of the challenges in providing estimations are:

- Having grey areas in the requirement: most of the agile team faces this challenge. In agile customers are not clear about their requirements, this becomes the most significant issue since the requirement of having lots of uncertainty.

- Epic level requirements: having a large requirement in one story often create difficulty when it comes to estimation. To produce more accurate estimation, it is vital to create subtask and divide the requirement in a meaningful way.
- Optimistic estimation: commonly the estimations are provided considering the ideal and optimistic situations, however in real life, there will be frequent requirement changes, unavailability of some resources, version mismatch can happen, because of this, the estimation provided can become wrong.
- Estimated by a single person: estimation should be provided considering all the members in the team since there will be different levels of experience people working in a project, an estimation provided by a senior person can be very small when the task comes to junior level person for development. In agile, poker card is a solution to overcome this kind of situations.
- Not considering buffer and dependencies: sometime developer provides tough estimation to prove them self, or because of the pressure they received from the project manager or product owner. Normally having a 15 to 20% buffer is a smart way to avoid a situation such as having an internal or external dependency, requirement changes etc.

Some solutions to mitigate the issues

- Ask clarification questions to clarify more requirements:
- Create many stories as much as possible from an epic:
- Estimation as a team:
- Having a proper buffer:

### **2.3. Challenges in Measuring the Productivity**

According to Scacchi (1995), development team productivity is to be calculated to reduce the software development costs, to improve the quality of deliverables that been produced, and to increase the rate at which software is to be developed. According to him, the software productivity is to be measured to recognize the top performers to reward and identify the bottom performers to provide the training.

The major productivity improvements can result in a substantial amount of savings in development costs (Scacchi, 1995). Measuring productivity helps in identifying underutilized resources (Nwelih & Amadin, 2008). The study of software productivity is important because higher productivity leads to lower costs (Bouchaib & Charboneau, 2005).

Bouchaib and Charboneau (2005) have studied the comparison of productivity of in-house developed projects and productivity of outsourced projects to a third party with a sample of 1085 projects implemented worldwide. Krishnan, Kriebel, Kekre and Mukhopadhyay (1999) have studied the software life cycle and productivity, which includes both maintenance and development costs and drivers of software team productivity and quality such as personnel capability, product size, usage of tools and software process factors. According to Banker and Kauffman (1991), software products can be found from the following formula.

### **2.3.1. Performance measurements of value creation are missing**

No measurements of value created or value to be created were identified. When asked about value creation a typical response was that it is difficult to demonstrate the value of a new product that is the incremental development and replacement of a product already in the market. Even if the interviewees indicate that a value perspective is needed and valuable, it is very difficult to define the metrics to capture the value of the development effort. Still, all of the five case companies have a structured process to develop a clear business case to initiate a development project. This information is used to gather internal funding for the project. The same regards to post-project evaluations; these evaluations focus on evaluating project execution proficiency regarding time, cost and quality, and not in terms of value created. (Michael A. Cusumano, Chris F. Kemerer, 1990)

### **2.3.2. No productivity measurements on an r&d level are found**

The concept of productivity as input divided by output is not measured on an R&D level. Instead, the focus of the performance measurement system is mainly on the cost and time perspective, i.e. the denominator not on the numerator of productivity. The

resource consumption part is prioritized while the gain or the result of the effort is missing in the measurements. A typical response, when asked about productivity, is that it would be interesting to have, to balance the perspective of cost, time, and quality, with the value created. (Michael A. Cusumano, Chris F. Kemerer,1990)

### **2.3.3. Performance measurement process is missing**

Many companies do not have a defined process for managing software developer's performance measurements. This case company was using a process based on the ISO/IEC standard, a software engineering and software measurement process. Organizations use an ad hoc process, very much dependent on the individual manager. As some managers expressed it, we have improved our measurements a lot during the last five years; we measure things like mean time between failures (MTBF), delays, time adherence, project cost, product quality, etc. They are fairly good measurements, but the difficult thing is what to do with the information. (Michael A. Cusumano, Chris F. Kemerer,1990)

## **2.4. Agile Software Development Methodology**

Agile software development methodology has taken the software development industry by storm and rapidly cemented its place as “the gold standard.” Agile methodologies are started based on four core principles as mentioned in the Agile Manifesto.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

These development methodologies are very much rooted in adaptive planning, early delivery as well as continuous improvement, all about respond to change quickly and easily. As a result, it is no wonder that 88% of responses in VersionOne's 2017 was mentioned that “ability to adapt to change” as the number one benefit of embracing Agile. (Agilemethodology.org, 2016)

### **2.4.1. Scrum framework**

Scrum is an iterative and incremental based software development method driven by the Product Backlog, which contains all active product requirements. The Product Backlog is managed by Product Owner, who is the only person authorized to change priorities of the requirements. (Scrum Alliance, 2016)

Scrum structures product development in cycles of work called Sprints, iterations of work which are typically 1- 4 weeks in length. Each Sprint is initiated with a Sprint planning meeting, where the Sprint Backlog is formed. Sprint Backlog is considered as a subset of Product Backlog requirements that defines the function which needs to be developed in the current Sprint. Every requirement can be further broken into tasks.

Functionality is developed by the software development team, i.e. a group of software developers that are together responsible for the success of each iteration, and of the project as a whole. Teams are self-managing, self-organizing, and cross-functional, and they are responsible for figuring out how to turn Product Backlog into an increment of functionality within the Sprint. (Scrum Alliance, 2016)

The ScrumMaster is responsible for conducting the Scrum process in the company so that it fits within an organization's culture and still delivers the expected benefits, and for ensuring that everyone follows Scrum rules and practices. The ScrumMaster facilitates a 15-minute daily Scrum meeting where every team member answers the three questions: "What they have done on this project since the last daily Scrum meeting?", "What they will do before the next meeting?" and "Do they have any obstacles?" The ScrumMaster is also responsible for resolving impediments encountered during the Sprint to assure the running smooth process flow.

At the end of each sprint, a sprint review meeting is held at which the team presents results produced in the sprint to the Product Owner. After the Sprint review and before the next Sprint planning meeting, the ScrumMaster also holds a Sprint retrospective meeting to ensure continuous improvement. (Scrum Alliance, 2016)

#### **2.4.1.1. Sprint**

A Sprint is a period when a team is focusing on meeting the Sprint commitments. During this period the team is supposed to have full authority over its own actions and no external influence by Product Owner, or anybody else is allowed. (Scrum Alliance, 2016)

Each Sprint has two elements, the Sprint goal and the Sprint Backlog. The Sprint goal is a relatively high-level description of a high priority item of the Product Backlog. It is an objective that will be met through the implementation of the Product Backlog. After establishing the overall Sprint goal, the team works with the Product Owner to determine the work required to reach the goal. Generally, a Sprint lasts for thirty calendar days. (Agilemethodology.org, 2016)

#### **2.4.1.2. Story point estimation**

A story point is a unit to measure the effort of a User Story or a feature. A point is assigned to each an every user story. These Points are relative in nature, i.e. a story that is assigned with a two-point value is considered to take twice the effort compared to the story that is assigned with a single point value. A Story Point is assigned based on the effort needed, the complexity and the inherent risk in developing a feature.

To estimate a user story, it requires some previous experience performing estimating, to have access to old historical data and have the freedom to use a trial based estimation approach if required.

To aid estimation, an expert may be asked about how long it will take to achieve the desired goal. The expert may rely on his/her intuition or previous experience. The benefit of using expert opinion is that it is not time-consuming. However, this method is not beneficial in an agile environment as here estimates are assigned to user-valued functionality which requires domain knowledge of different members working in the team. This makes it difficult to find suitable subject matter experts in different disciplines to evaluate the work effort. Alternatively, the user stories can be estimated



against already estimated user stories. There is no need to compare all the stories against a single baseline or common reference. (Scrum Alliance, 2016)

A Story can be disaggregated into smaller, easier to estimate blocks. However, there is no safety check when disaggregating a user story. The likelihood of missing out a story increases with disaggregation. Summing up estimates of a number of minor tasks may further cause different issues. (Agilemethodology.org, 2016)

#### **2.4.2. Existing performance metric and KPIs**

This chapter contains a description of some of the available Matrices or KPIs which can be used to track or measure the productivity of the software development process in the agile environment.

##### **2.4.2.1. Burn down charts**

A burndown chart shows the team's progress toward the completion of all of the story points they agreed to complete in a sprint. This chart starts with the total points the team has to deliver on the sprint, and tracks on a day-to-day basis for how many of those points have been completed and are ready for the sprint demo. (Scrum Alliance, 2016)

The burndown charts are usually maintained by the scrum master and may be updated on a daily basis, perhaps after the daily stand up, or on a continuous basis if it is generated automatically by the tools which were used to maintain in the scrum board. The primary audience for a burndown chart is the team itself, although there may be story points on a burndown chart that could be relevant to people outside the scrum team.

A typical burndown chart starts with a straight diagonal line from the top left to the bottom right, showing an "ideal" burn down rates of the sprint. In general, the points are not to match with ideal points, but rather to keep in mind that how much of the sprint is left at any point of time, and how much effort that the team expects to be able to put toward the development of the product on any day of the sprint.

Lines or columns on the burndown chart may be used to represent the number of points (effort) remaining in the sprint. Starts with the number of points the team has committed to the planning. As work is completed, these columns should become lower until they touch the point zero.

Few teams have the approach to track the daily work completed, either in story points format or individual tasks toward sprint goal. This can be completed with a line or stacked columns, tracking these daily metrics towards the burndown chart so they can create more visibility of the performance.

There are few legitimate reasons for a column to be higher on one day than it was on the previous day. If a bug is identified before the end of the sprint, and a story that was marked as complete or ready to perform demo needs to be revisited on again, columns may increase in size over the days. New stories pushed into the sprint after the sprint has started may also become a reason as one day's column to be higher than the previous day's value. A pattern of rising columns on a burndown chart may indicate that the scope of the work is exceeding the originally agreed sprint backlog, which is an anti-pattern in the scrum. (Scrum Alliance, 2016)

#### **2.4.2.2. Velocity graph**

Velocity is how a scrum team measures the amount of work they should be able to complete in a typical sprint. Velocity is measured historically, from one sprint to the other. By tracking the story points the team should be able to finish according to their definition of done, they can build up to a level of reliable and predictable sense of how much of effort it will take the team to finish the new user stories based on their relative points. (Scrum Alliance, 2016)

Keeping track of velocity is the duty of the teams' scrum master. At the end of the sprint demo, the scrum master should be able to calculate the story points which were estimated for the user stories that were considered as completed during that sprint. This number should be added as an input data point on the velocity chart for that sprint.

Velocity chart tends to start out popping around from high numbers to low numbers, as the team learns how much effort they completed in one sprint, and how to estimate user stories. When a team works together, they became better to estimate stories relative to each other. This skill leads to a better sense of how many stories, and how many points, the team can accomplish in a single sprint.

Over time passes, if the composition of the team stays as consistent, the velocity chart that started as very erratic will start to find itself averaging toward a consistent value. Not like many charts in a business environment, the point of the velocity chart is not to see a continuous increment, but indicate the values is intersected around a consistent horizontal line. That line represents the amount of work effort the team can realistically and sustainably accomplish in a single sprint. (Scrum Alliance, 2016)

## **2.5. Summary**

This chapter reviewed the current literature available in related to productivity measurement of the software developer in an agile-based scrum environment. The definition of the term productivity in the context of software developer was discussed, challenges in measuring the productivity were identified, current KPIs and indicators about the productivity in the agile base scrum environment were discussed. In the next chapter research methodology, measures and measurements are described.

### **3. RESEARCH METHODOLOGY**

This Chapter presents the research methodology; it discusses the mix method approach to identify the factor which can be used to measure the software developer's productivity. Moreover, the process used to conduct the survey and the interviews.

Section 3.1 elaborates the research problem and the purpose of this research, section 3.1.1 contains details about the research method used for this research, section 3.1.2 contains information about the data collected through the initial survey as well as the interview conducted among the information technology professionals, section 3.1.3 details about the population and sample sections of the data which was collected for the purpose of this research, section 3.1.4 explains about the process which was followed to collect the data from the targeted group.

#### **3.1. Research Problem**

In the modern world, the computer has a vital part to play; modern technology keeps improving human life day by day to a better position. Software development is getting complex and at the same time organization are expecting faster deliveries from developers.

From small medium level organization to large enterprise level organization are looking for the best talents, developers who can produce best outputs within a short period.

Based on the organization culture, different organizations have different methods to evaluate the software engineer's productivity. However, the main objective of this research thesis is to find out a most suitable and common model to measure the productivity of the software engineers in the scrum-based agile environment.

Purpose of this research is to get a better understanding regarding the following areas

- Initially, the purpose of this research is to understand the team Productivity when it comes to software development.

- Identify the factors which can be considered to measure the productivity of software developers.
- Identify the opportunities for improvement based on the measurement output.

### **3.1.1. Research method**

Figure 2 elaborates the research methodology used for this research thesis. The research is conducted in quantitative methodology which involves in collecting, analyzing and integrating data through quantitative approaches such as a survey and data analysis. The research problem was identified based on the literature review as well as analysis of local agile projects.

Initially, based on the literature review most important factor which can indicate productivity was identified, in order to identify the relationship among the factors, software development activity related data was gathered from different team and data was analysed.

Finally, a survey questionnaire was created based on the literature review as well as data gathered from development activity to validate the understanding. Software engineering professionals were involved to provide the answers for the questionnaires. Later the result of the survey was analysed and evaluated.

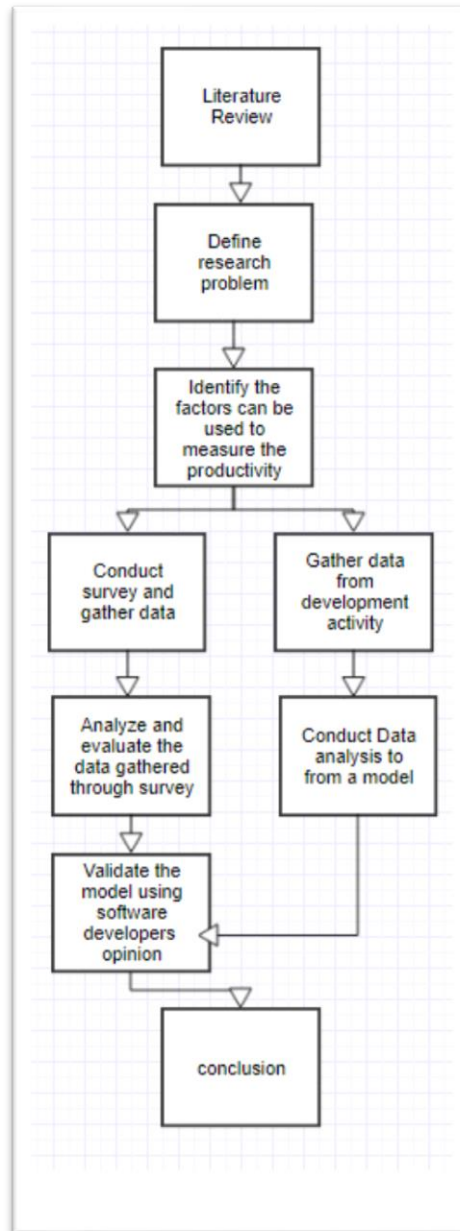


Figure 2 Research method

### 3.1.2. Conceptual framework of the research

The literature survey provides the basis and the foundation to develop a Conceptual Framework to explore the research problem in a more useful manner. Chapter III was dedicated to present the problem in an abstract form suggesting the hypothetical relationship between the main problem and the related variables.

The Operationalization is useful to empirically express the problem in the form of variables, indicators and measurements. The Conceptual Framework is thus the working model based on which the testable hypothesis would be generated.

The Table below represents the most contributing factors with their references which can be referred to measure the productivity of the software developer.

Table 1 represents the most common indicating factor in their references which can be used to understand the productivity of a software developer.

Table 1 Factors in the conceptual framework

<b>Factors</b>	<b>Reference</b>
Code Quality	G. P. Sudhakar (2012) Dana T. Edberg & Brent J. Bowman (1996)
Code Complexity	Amel Ben Hadj Salem Mhamdia (2013) Simonetta Balsamo (2014)
Code Quantity	H.C. Shiva Prasad Damodar Suar (2010) Adam Trendowicz (2009)
Work Effort	Chris Peck (2002) Dale W. Callahan (2002)
Software developer productivity	Adrián Hernández-López (2015) Inga Podjavo (2017)

The section below is dedicated to providing definitions of the key concepts depicted in the Conceptual Framework.

**Code Quality:** A software developer can take many different approaches to develop a single feature. Some developers may try to finish the work using a shortcut and others sometimes unnecessary frameworks or workaround to complete the work. However, there should be a predefined or benchmarked coding standard which all the team members should follow to increase the maintainability and security and performance of the software product. Therefore, the purpose of this variable is to represent employee willingness to accept code quality as a software productivity measurement factor.

**Code Complexity:** A smart or experienced developers can produce complex code and complete a task without creating many classes and lengthy code. Therefore, the purpose of this variable is to represent employee willingness to accept code complexity as a software productivity measurement factor.

**Code Quantity:** An average software engineer can produce remarkably more LOC per unit time than is possible, Therefore, the purpose of this variable is to represent employee willingness to accept code quantity as a software productivity measurement factor.

**Work Effort:** In information technology industry completing the project on time is very essential, produce a high-quality product within a given time work is a challenge where most of the software engineering professional facing. Therefore, the purpose of this variable is to represent employee willingness to accept Work effort measurement as a software productivity measurement factor.

**Software Developer Productivity:** The purpose of this variable to identify the productivity score of a software engineer for a sprint.



### 3.2. Development of Hypotheses

Table 2 represents the list of Hypotheses which were developed based on the Literature review in Chapter 2 and the Conceptual model depicted in Section 4 of Chapter 5.

Table 2 Research hypotheses

Alternative Hypothesis	Null Hypothesis
<b>H1a:</b> There is a positive correlation between code quality and the productivity of a software developers	<b>H1o:</b> There is no positive correlation between code quality and the productivity of a software developers
<b>H2a:</b> There is a positive correlation between code complexity and the productivity of a software developers	<b>H2o:</b> There is no positive correlation between code complexity and the productivity of a software developers
<b>H3a:</b> There is a positive correlation between code Quantity and the productivity of software developers.	<b>H3o:</b> There is no positive correlation between code Quantity and the productivity of a software developers
<b>H4a:</b> There is a positive correlation between minimum work effort and productivity of software developers.	<b>H4o:</b> There is no positive correlation between minimum work effort and productivity of software developers.

### 3.3. Operationalization

The key variables, indicators and measures used in the research study are indicated on the operationalisation Table 3.

Table 3 Operationalization

Variable	Indicator	Measurement (5 Points Likert Scale)	KPI
Code Quality	The low number of bugs	Questionnaire	Q1

	Adherence to pre-defined code standards	Questionnaire	Q2
	High score in code maintainability	Questionnaire	Q3
	Successful code reviews	Questionnaire	Q4
Code Complexity	Lines of code in a class	Questionnaire	Q6
	Average lines of code in a function	Questionnaire	Q7
Code Quantity	High cyclomatic complexity within the threshold	Questionnaire	Q9
	High module design complexity within the threshold	Questionnaire	Q10
Work effort	Total hours required to complete the task	Questionnaire	Q12
	Story points	Questionnaire	Q13
	Impact of Code quality	Questionnaire	Q5
	Impact of code Complexity	Questionnaire	Q8
	Impact of code Quantity	Questionnaire	Q11
	Impact of work effort	Questionnaire	Q14

Definition of 5 points Likert scale

- Strongly Agree - 5
- Agree - 4
- Neutral - 3
- Disagree - 2
- Strongly Disagree – 1

### **3.3.1. Population and sample selection**

Five different team's two sprint development activity related data is gathered for the purpose to form a model to measure the productivity.

According to the National ICT Workforce Survey 2010, there are about 11013 employees working as software development professionals in software engineering organizations.

Figure 3 depicts the categorization and the count of each categorization of the ICT workforce in Sri Lanka.

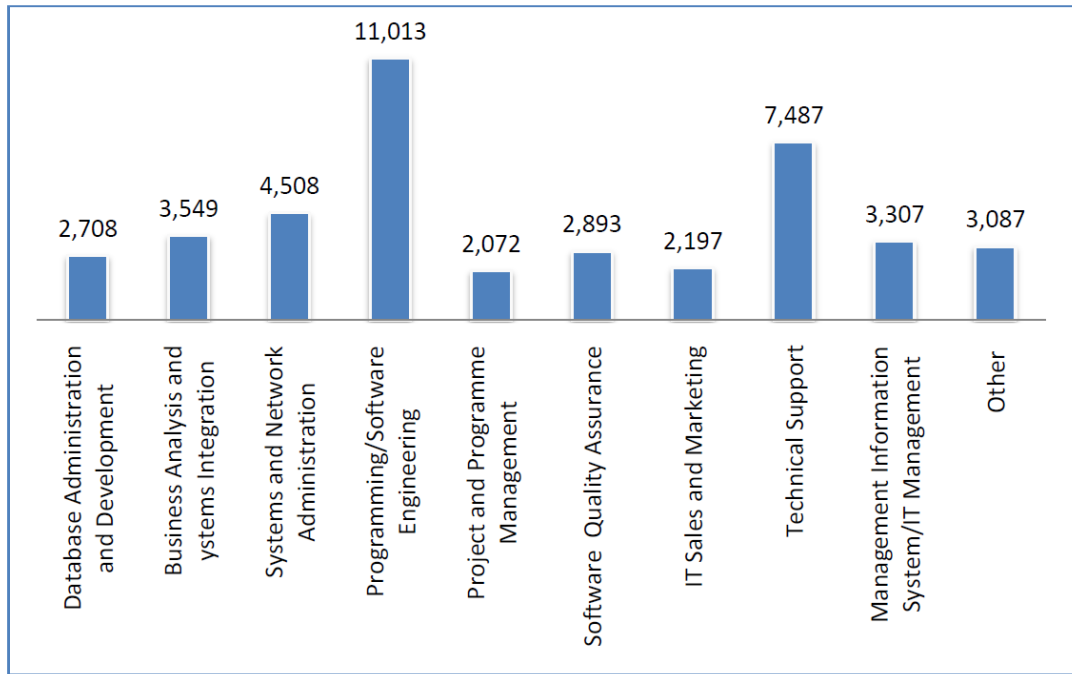


Figure 3 Overall IT workforce by job category

Following numbers were obtained from the “National ICT Workforce Survey 2010” by SLASSCOM.

$$X = Z \left( \frac{C}{100} \right)^2 r (100 - r)$$

$$n = \frac{N x}{((N - 1)E^2 + x)}$$

$$E = \sqrt{\frac{(N - n)x}{n(N - 1)}}$$

- The margin of error is 5%
- The confidence level is 70%
- Population Size is 11013
- n Sample Size
- N is the population size
- r is the fraction of responses

- $Z(c/100)$  is considered as the critical value for the confidence level  $c$

Hence, the recommended sample size for this research survey is: 107 software developers

### **3.3.2. The process of data collection**

A quantitative approach is followed in conducting the research. Initially Development activity related data was gathered from the project management tool such as JIRA and Confluence.

Secondly, a survey is conducted to gather information from the targeted group of software developers. The survey was conducted using Google form to gather information, as a survey results total of 109 responses were recorded.

The main variables which will be used to test the Hypothesis are:

- Code Quality
- Code Complexity
- Code Quantity
- Work Effort

Minitab and SPSS are used to discover a correlation between variables. Microsoft Excel is used to present data collected under demographic and general information section of the questionnaires where the data is divided into categories such as age, gender, experience etc. Furthermore, a different type of charts such as pie charts and bar charts are used to represent the gathered data.

Figure 4 depicts the Process of data collection followed during this research thesis.

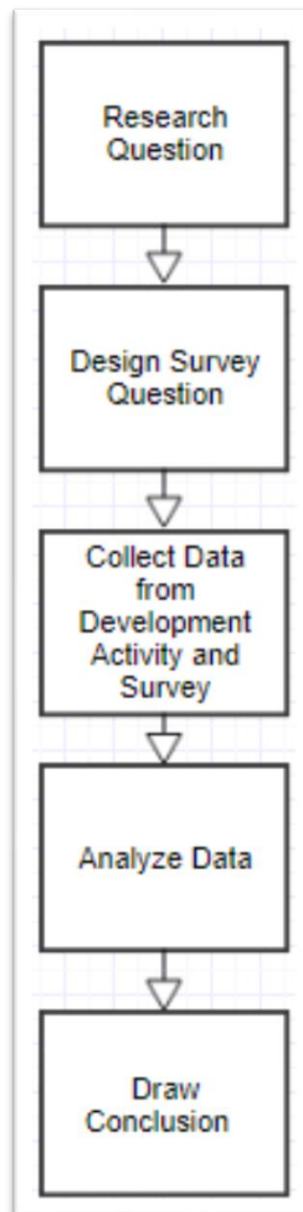


Figure 4 Process of data collection

### 3.4. Summary

This chapter presented the research methodology used in the study. Four most essential indicators were found out through literature review, and those are tested in this study. Baseline values defined by the company and industry standards are used to measure the IT service quality. As a second phase, six different software development data such as sprint data, bugs reported, story point estimation, actual hours taken to complete the stories, bugs etc. were collected for the purpose to validate and fine-tune the model.

## 4. DATA ANALYSIS

### 4.1. Introduction

This chapter presents an analysis of the data gathered for the research study through the project management tools such as JIRA, software source code version controlling tool such as Git and as well as through a questionnaire. The analysis of the data consists validation of research data and instrument. Furthermore, the analysis of data also consists description of data transformation to derive further data which needed to formulate the analytic model to measure the productivity of the software engineers.

### 4.2. Data Gathered from Software Development Activity

Software project related data was gathered from JIRA which is a software project management, Git which is a version control tools for software source code.

Five different software development projects from a well-established software engineering company is selected for the data gathering; there is around 140 employees are working in 26 different scrum-based agile projects in the particular organisation.

From the project management tool following information was exported for the analysis purposes.

1. Bugs recorded
  - a. Type of bug
  - b. The effort took to fix the bug
  - c. Associated story
2. Stories allocated for the sprint
  - a. Sprint that the story is allocated to
  - b. Effort estimated for the story
  - c. Effort spent on the story
  - d. Bugs associated with the story
  - e. Source code commit id in Git

Following information exported from the source code version controlling tool

1. Code Committed
  - a. Lines of code

- b. Code complexity
- c. Class and function created

### 4.3. Descriptive Statistics for Development Activity Data

A descriptive analysis was done for the demographic data to analyze the data gathered regarding their age category, gender and experience of the software engineers.

#### 4.3.1. Development activity data by age

Below figure depicts the breakdown by age group of the software engineers in the sample data gathered from the software development teams.

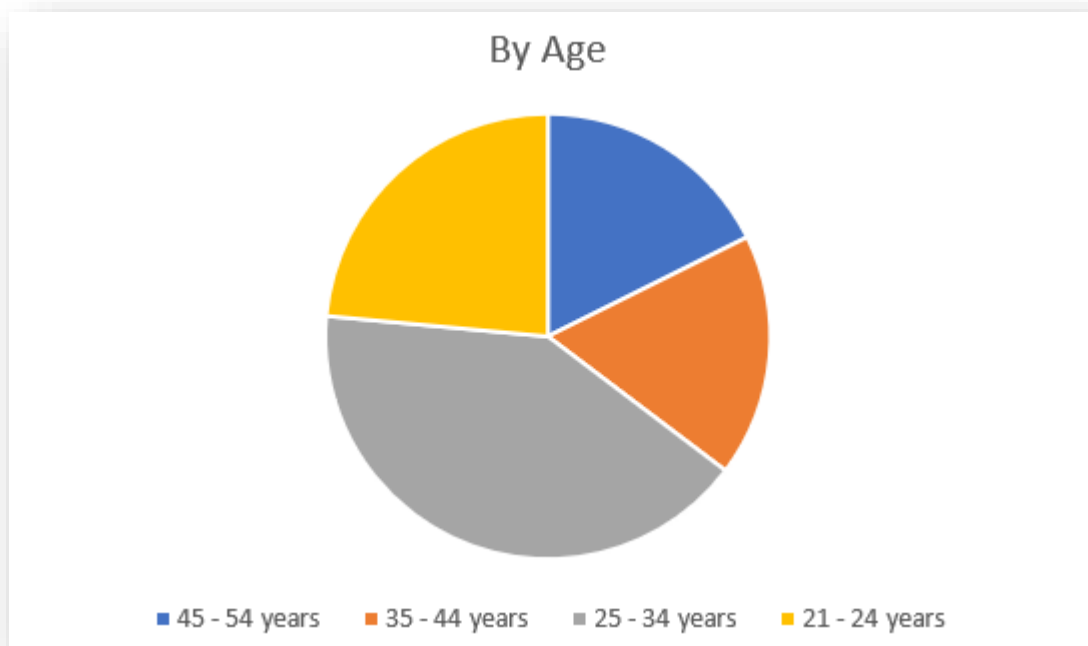


Figure 5 – Software development team sample grouped by age

Approximately 41. % of the software engineers were between the age of 25 and 34, 23% of the software engineers were between the age of 21 and 24, both groups of 35 to 44 and 45 to 54 have each has 18% value in the sample data.



#### 4.3.2. Sample of software engineers categorized by experience level

Below figure depicts the breakdown by the experience level of the software engineers in the sample data gathered from the software development teams.

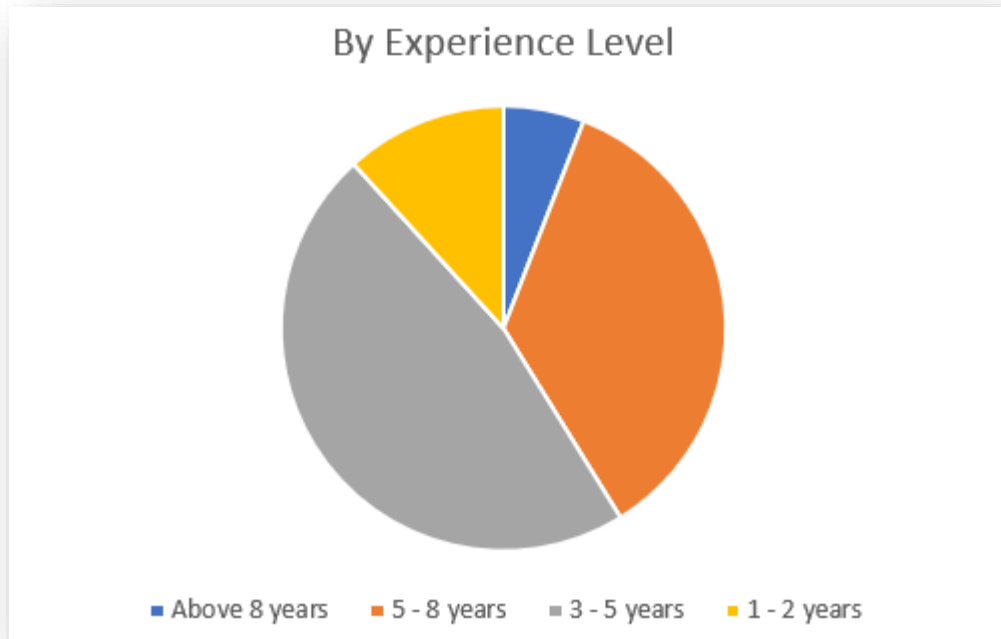


Figure 6 - A sample of software engineers categorised by experience level

Approximately 47% of software engineers were having 3 -5 years of experience, 35 % have 5 - 8 years of experience, 12 % is having 1 -2 years of experience and Nearly 6 % of the software engineers are having more than eight years of working experience.

### 4.3.3. Sample data of software engineers categorized by gender

Below figure depicts the breakdown by gender who responded to the survey questionnaire regarding their age category.

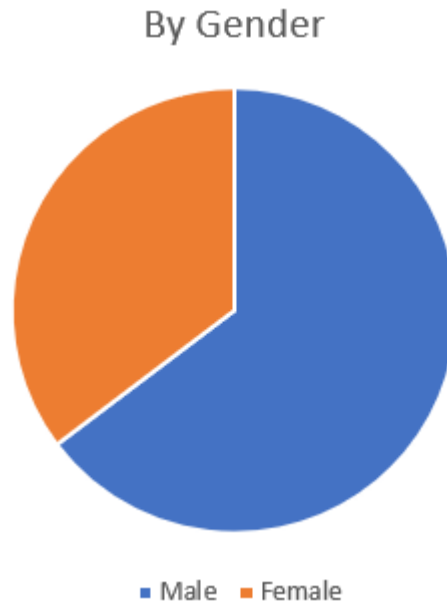


Figure 7 - A Sample Data of Software Engineers Categorized by Gender

Approximately 65% of the software engineers were Males whereas 35% of the software engineers were Females.

### 4.4. Extraction and transformation of development activity data

Development activity related data was gathered from a project management tools such as JIRA and confluences then transformed to required data format using a suitable formula.

Below mentioned formulas are used to extract code quality, code complexity, code quantity and actual work effort related data from development activity information.

#### 4.4.1. Code quantity

Quantity is a most straightforward and fundamental factor which falls under the developer's output category Since the metric for code quantity is simply a count of lines of code in a function, class or program.

By analyzing the code commits of different developers, there are scenarios where the developer has committed which are not the same programming language as a result of their development work. Since counting the line of code as a measurement of quantity will not give the best result as an output. This is due to the difference in high and low-level languages. While a function in a high-level language may take five lines, the same function in a lower level language may take fifteen lines to complete.

As a solution language consideration must be factored in when using a line of code measure across multiple languages. It is possible to adjust for language differences by dividing a Lines of Code by a language adjuster such as average lines of code per function. For our purposes, the quantity factor of the programmer productivity equation will be calculated at the class level and adjusted for language. Lines of code will be calculated excluding comments and blank lines. This exclusion will render a standard for comparison regardless of commenting and spacing style. The lines of code metric will then be divided by the average lines per function for the code language. The result of this equation is a language-weighted quantity measure roughly equal to the number of functions (Peck, C., & Callahan, D.) 2008.

$$\text{Quantity} = \frac{\text{LOC}}{\text{Average Lines per Function for the code}}$$

#### **4.4.2. Code quality**

Producing quality software is part of the responsibility of a software programmer. By nature of this responsibility, there is an implied prorating of output based upon quality. A poor-quality piece of code should not be considered the same as a high-quality piece of similar size. This is reflected in the quality component of the Productivity equation. The following are metrics that will contribute to our calculation of software quality. Defects per lines of code is a standard way to measure software quality. This metric describes the rate at which errors inside code have been uncovered. Again, language plays an important part in interpreting Defects per Lines of Code. A high-level language is more likely to have a higher defect per line rate than a lower level language.

This is simply due to the number of lines that it takes to perform a function. It is possible to adjust for language differences by dividing Defects per Lines of Code by a language adjuster such as average lines of code per function. The result is a defect rate, adjusted for language. Finding and dealing with defects is dependent upon the methodology of each organization. The proposed study is dependent upon several methodology practices that are present at the test site.

The two most important in reference to quality measures are code reviews and the Software Quality Assurance process. Before a program is submitted to SQA, the code is submitted for a peer review. Defects uncovered during this peer review are reported, tracked in a central repository and fixed before submission to the SQA department. Below formula can be used to identify the code quality of a story (Peck, C., & Callahan, D.) 2002.

$$\begin{aligned}
 \text{Code Quality} = 1 - & \left( \left( \frac{\text{Number of Critical bugs}}{\text{Adjusted Lines of Code}} \right) * \text{Fatal Severity Weight} \right) \\
 & + \left( \left( \frac{\text{Number of Major bugs}}{\text{Adjusted Lines of Code}} \right) * \text{Major Severity Weight} \right) \\
 & + \left( \left( \frac{\text{Number of Minor bugs}}{\text{Adjusted Lines of Code}} \right) * \text{Minor Severity Weight} \right)
 \end{aligned}$$

Following weight was considered as severity weight for following types of bugs

1. Critical bug weight = 5
2. Major bugs weight = 3
3. Minor bugs weight = 2

#### 4.4.3. Code complexity

The proper calculation of complexity factor will be determined through multiple comparisons of individual programmer's work within a close time frame. By keeping the time frame close, a programmer's productivity should not change drastically. Therefore, different completed and tested classes made by a programmer can be compared to determine if the complexity factor is reflecting and predicting productivity.

McCabe metrics were used to measure the structural complexity of the code. The measurement is based on the complexity of the logical path within a function, to reduce the complexity following two McCabe metrics were used to determine the complexity of each function within the code.

Cyclomatic Complexity – Number of linearly independent paths within a function. Automated tools can be used to calculate the complexity considering above matrices; the matrices will then be used to formulate a complexity factor for the code produced. The matrices above will deliver the complexity at a functional level since it is vital to creating the average complexity at the class level.

Following a weighted average complexity, the metric could be used to calculate the weight average class metric complexity (Peck, C., & Callahan, D.) 2002.

$$\begin{aligned} & \textit{Weighted Average Class Complexity Metric} \\ &= \frac{(\textit{Function1 Metric}) * (\textit{Function1 LOC})}{\textit{Total LOC}} \\ &+ \frac{(\textit{Function2 Metric}) * (\textit{Function2 LOC})}{\textit{Total LOC}} + \dots \\ &+ \frac{(\textit{FunctionN Metric}) * (\textit{FunctionN LOC})}{\textit{Total LOC}} \\ & \textit{Complexity Metric} = \frac{(\textit{Weighted Average Class Complexity Metric 1} + \\ & \textit{Weighted Average Class Complexity Metric 2} + \\ & \dots + \textit{Weighted Average Class Complexity Metric N})}{\textit{Number of classes created for the story}} \end{aligned}$$

Through this equation, average complexity is calculated, the entire complexity of the class should be able to determine.

#### **4.4.4. Work effort**

Actual hours taken to complete all the stories allocated for the particular developer is considered as the work effort for the sprint.

#### **4.4.5. Productivity**

The productivity of a software developer who is working in the scrum-based agile environment is calculated by using following formula.

$$Productivity = \frac{Work\ allocated\ for\ the\ developer}{'s\ actual\ Work\ Completed}$$

While performing capacity planning following formula is used to calculate the work allocation for a developer.

*Work allocation*

$$= Number\ days\ in\ the\ sprint * Working\ hours\ per\ day \\ * Focus\ factor$$

Focus Factor is a developer's ability to remained focused on the sprint goal without getting distracted. As a norm following values are considered as focus factor

1. An average developer who is already working in the team more than average = 0.8
2. Newly joined developer = 0.4

#### **4.5. Testing Hypothesis - Pearson's Correlation Analysis**

According to the obtained development activity data, correlation analysis is used to identify the strength of the relationship between the variables. The following output is obtained using SPSS. To determine the connection between variables, correlation analysis was done. Standard averaging had been used for each variable to analyze the

significance, by using Pearson Correlation Matrix. Correlation value “r” was defined as follows:

- 0.80 or higher - Very strong relationship
- 0.60 to 0.79 - Strong relationship
- 0.40 to 0.59 - Moderate relationship
- 0.20 to 0.39 - Weak relationship
- to 0.19 – Very weak relationship

Significant value denotes the probability of correlation occurrence and a significant value less than 0.01 (1%) was considered as significant.

#### 4.5.1. The correlation between code quality and productivity

Below figure presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.731 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

Correlations			
		Productivity	Quality
Productivity	Pearson Correlation	1	.731**
	Sig. (2-tailed)		.000
	N	36	36
Quality	Pearson Correlation	.731**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 8 - The correlation between Code quality and productivity

It is indicating that code quality is an important factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H1a is justified, and therefore the null hypothesis H1o is rejected.

#### 4.5.2. The correlation between code quantity and productivity

Below figure presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.714 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

		Productivity	Quantity
Productivity	Pearson Correlation	1	.714**
	Sig. (2-tailed)		.000
	N	36	36
Quantity	Pearson Correlation	.714**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 9 - The correlation between Code quantity and productivity

It is indicating that code Quantity is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H2a is justified, and therefore the null hypothesis H2o is rejected.

#### 4.5.3. The correlation between code complexity and productivity

Below figure presents the two-tailed person correlation result for code complexity vs productivity of software developers. The value of the Pearson Correlation Coefficient at 0.693 which is significant at the 0.01 level indicates that there is a **positive relationship** between the two variables.



### Correlations

		Productivity	Complexity
Productivity	Pearson Correlation	1	.687**
	Sig. (2-tailed)		.000
	N	36	36
Complexity	Pearson Correlation	.687**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 10 - Correlation between code complexity and productivity

It is indicating that code Complexity is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H3a is justified, and therefore the null hypothesis H3o is rejected.

#### 4.5.4. The correlation between actual hours worked and productivity of a software developer

Below figure presents the two-tailed person correlation result for minimal work effort vs productivity of software developers. The value of the Pearson Correlation Coefficient at -0.747 which is significant at the 0.01 level indicates that there is a **strong negative relationship** between the two variables.

### Correlations

		Productivity	Hours Worked
Productivity	Pearson Correlation	1	-.747**
	Sig. (2-tailed)		.000
	N	36	36
Hours Worked	Pearson Correlation	-.747**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 11 -Correlation between minimal work effort and productivity

It is indicating that code quality is an important factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H4a is justified, and therefore the null hypothesis H4o is rejected.

**4.5.5. The logarithm value of correlation between code quality and productivity**

Below figure presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.745 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

		LogProductivity	LogQuality
LogProductivity	Pearson Correlation	1	.745**
	Sig. (2-tailed)		.000
	N	36	36
LogQuality	Pearson Correlation	.745**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 12 -The correlation between Code quality and productivity

It is indicating that code quality is an essential factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H1a is justified, and therefore the null hypothesis H1o is rejected.

As the result of the above test, the following model can be driven

$$\text{Developer's Productivity} = \text{Code Quality}$$

Value for “a” should be able to drive from a regression analysis.

#### 4.5.6. The logarithm value of correlation between code quantity and productivity

Below figure presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.716 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

		LogProductivity	LogQuantity
LogProductivity	Pearson Correlation	1	.716**
	Sig. (2-tailed)		.000
	N	36	36
LogQuantity	Pearson Correlation	.716**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 13 - The correlation between Code quantity and productivity

It is indicating that code Quantity is an important factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H2a is justified, and therefore the null hypothesis H2o is rejected.

As the result of the above two correlation tests, the following model can be driven.

$$\text{Developer's Productivity} = \text{Code Quality}^a * \text{Code Quantity}^b$$

Value for “b” should be able to drive from a regression analysis.

#### 4.5.7. The logarithm value of correlation between code complexity and productivity

Below figure presents the two-tailed person correlation result for code complexity vs productivity of software developers. The value of the Pearson Correlation Coefficient at 0.705 which is significant at the 0.01 level indicates that there is a **positive relationship** between the two variables.

		LogProductivit y	LogComplexit y
LogProductivity	Pearson Correlation	1	.705**
	Sig. (2-tailed)		.000
	N	36	36
LogComplexity	Pearson Correlation	.705**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

Figure 14 - Correlation between code complexity and productivity

It is indicating that code Complexity is an important factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H3a is justified, and therefore the null hypothesis H3o is rejected.

As the result of the above three correlation tests, the following model can be driven.

$$\text{Developer's Productivity} = \text{Code Quality}^a * \text{Code Quantity}^b * \text{Code Complexity}^d$$

Value for “d” should be able to drive from a regression analysis.

#### 4.5.8. The logarithm value of correlation between actual hours worked and productivity of a software developer

Below figure presents the two-tailed person correlation result for minimal work effort vs productivity of software developers. The value of the Pearson Correlation Coefficient at - 0.752 which is significant at the 0.01 level indicates that there is a **strong negative relationship** between the two variables.

**Correlations**

		LogProductivity	LogEffort
LogProductivity	Pearson Correlation	1	-.752**
	Sig. (2-tailed)		.000
	N	36	36
LogEffort	Pearson Correlation	-.752**	1
	Sig. (2-tailed)	.000	
	N	36	36

\*\*. Correlation is significant at the 0.01 level (2-tailed).

Figure 15 - Correlation between minimal work effort and productivity

It is indicating that code quality is an important factor which should be considered to get an idea about the software developer’s productivity. Based on the Pearson correlation value the alternative hypothesis H4a is justified, and therefore the null hypothesis H4o is rejected.

As the result of the above four correlation tests, the following model can be driven.

Developer’s Productivity

$$= \frac{\text{Code Quantity} * \text{Code Quality} * \text{Code Complexity}^c}{\text{Actual Hours Worke}^d}$$

Value for “d” should be able to drive from a regression analysis.

#### 4.6. Linear Regression Analysis

Initially, the data is explored to identify the linear relationship that might exist between the productivity and independent variables. Secondly, the data is also explored to identify the non- linear relationship that might exist between the productivity and independent variables.

As the result logarithm-based correlation values are higher comparing to non- logarithm-based correlation values, it is clear that the relationship between the productivity and the independent variable is a non- linear relationship.

As a next step, to identify the coefficients it is required to perform a regression analysis since the input variables/data is in the form of the logarithm, linear regression analysis is performed to analyze the coefficient.

The regression coefficient for the association between productivity and Quality, Quantity, Complexity and actual hours worked are shown below. The constant (intercept) is avoided as the constant value which is 0.085 was not meeting the significant requirement since the significant value is 0.05.

Model	Unstandardized Coefficients		Sig.	
	B	Std. Error		
1	(Constant)	-1.231	.693	.085
	LogQuality	.312	.120	.014**
	LogQuantity	.690	.192	.001**
	LogComplexity	.017	.026	.031**
	LogEffort	-.183	.073	.018**

a. Dependent Variable: LogProductivity , \*\*Correlation is significant at the 0.05 level.

Figure 16 - Non-Liner Regression Analysis

End of the above test productivity model can be formulated as below since value for the “a”, “b”, “c” and “d” was identified.

Developer’s Productivity

$$= \frac{\text{Code Quantity}^{0.690} * \text{Code Quality}^{0.13} * \text{Code Complexity}^{0.17}}{\text{Actual Hours Worked}^{0.183}}$$

**Model Summary<sup>b</sup>**

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.884 <sup>a</sup>	.782	.754	.02764

a. Predictors: (Constant), LogEffort, LogQuantity, LogQuality, LogComplexity

b. Dependent Variable: LogProductivity

Figure 17 Model Summary

Since approximately 75% of the variation in the response is explained by the model, it confirms the validity of the model.

**4.7. Reliability of Survey Data**

The reliability of the questionnaire used to collect the survey dataset was tested using Cronbach’s Alpha Coefficient (CAC). According to the analysis of CAC, internal consistency of question sets for each of the factors identified and used in the theoretical framework was found to be in the acceptable value range and are listed in table 4.

Table 4 Reliability of surveys data

Factor Category	Cronbach's Alpha ( $\alpha$ )
Quality	0.7672
Quantity	0.6948

<b>Code Complexity</b>	0.6568
<b>Work Effort</b>	0.6409
<b>The success rate of the Performance Appraisal System</b>	0.6868

#### 4.8. Descriptive Statistics for Survey Demographic Data

A descriptive analysis was done for the demographic data to analyze the respondents regarding their age category, gender and experience.

##### 4.8.1. Sample of software engineers grouped by age

Below figure depicts the breakdown by age group who responded to the survey questionnaire regarding their age category.

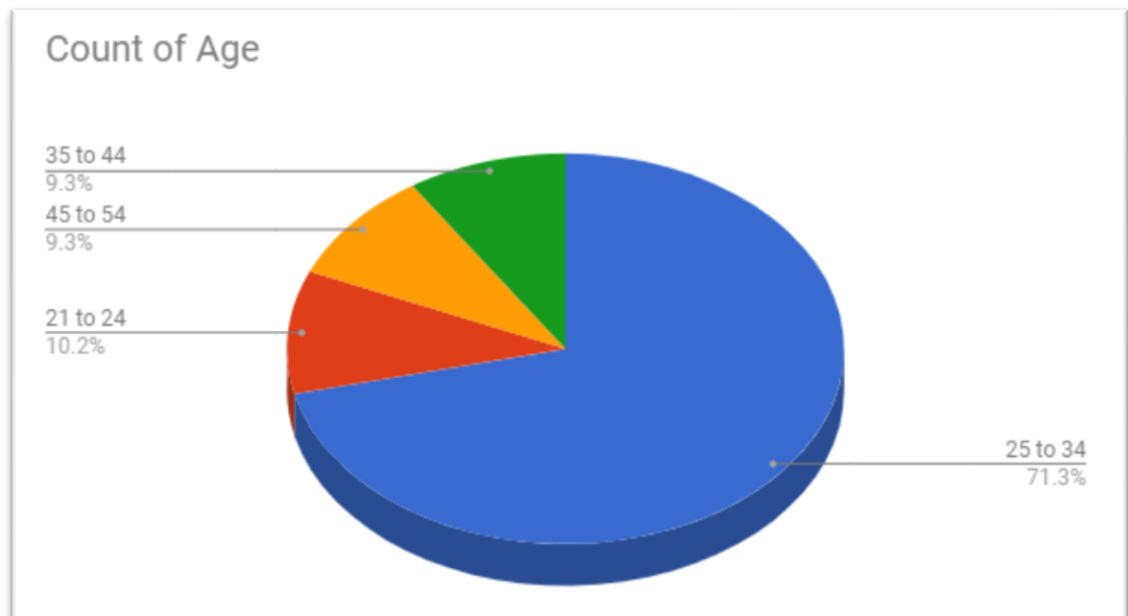


Figure 18 Sample of software engineers grouped by age

Approximately 10.2% of the respondents were between the age of 20 and 24, 71.3% of the respondents were between the age of 25 and 34, 9.63% of the respondents were between the age of 35 to 44. The percentage of respondents who were greater than 44 years of age was 9.3% of the overall sample. There are very young professionals working in the IT industry, and this fact was evident from the sample as well.



#### 4.8.2. Sample of software engineers categorized by gender

Below figure depicts the breakdown by gender who responded to the survey questionnaire regarding their age category.

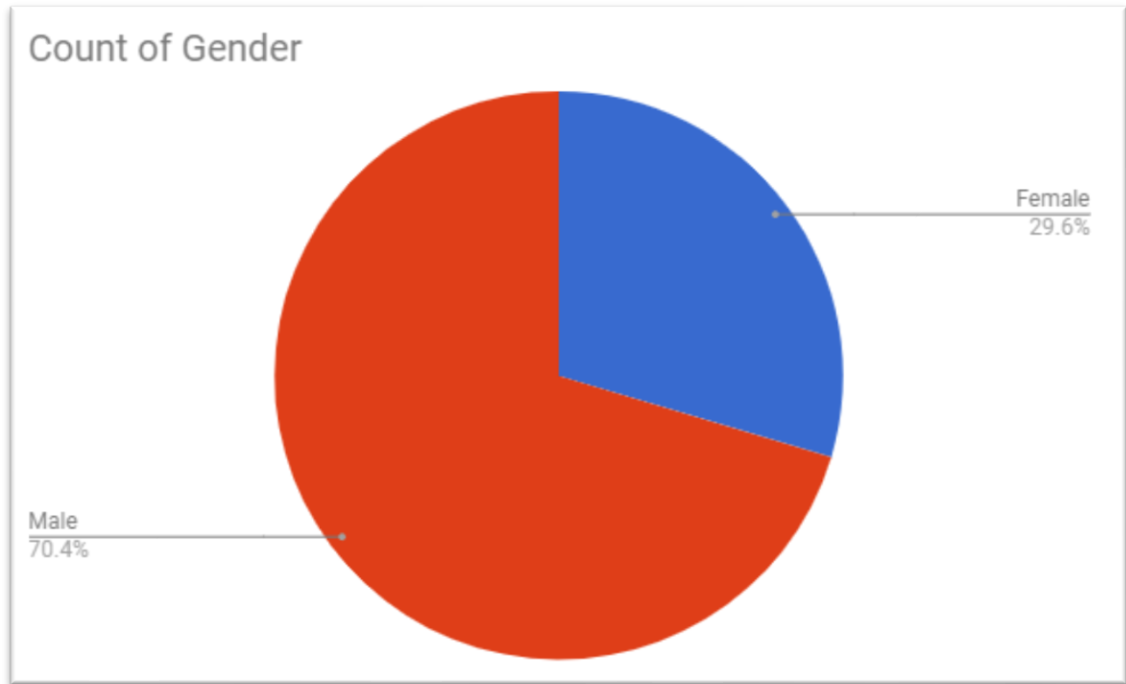


Figure 19 Sample of software engineers categorized by gender

Approximately 70.4% of the respondents were Males whereas 29.6% of the respondents were Females.

### 4.8.3. Sample of software engineers categorized by experience level

Below figure depicts the breakdown by experience level who responded to the survey questionnaire regarding their age category.

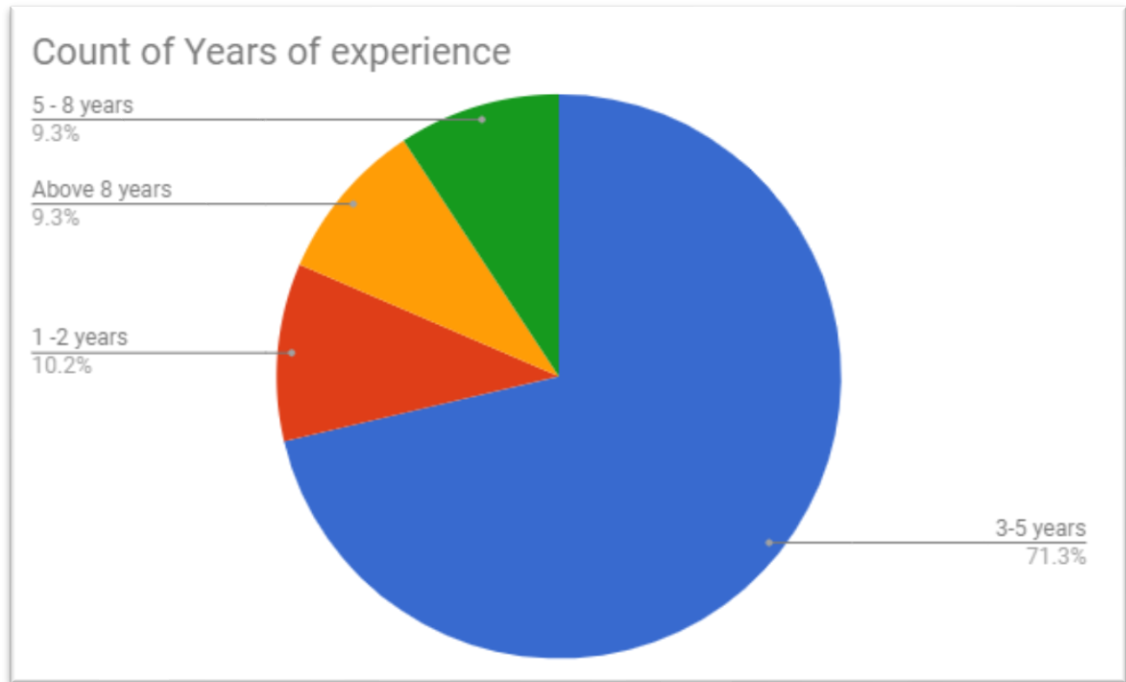


Figure 20 Sample of software engineers categorized by experience level

Approximately 10.2% of respondents are having 1-2 years of experience, 71.3% having 3-5 years of experience, 9.3% is having 5-8 years of experience, and Nearly 9.3% of the respondents are having more than eight years of working experience.

## 4.9. Presentation of Variable Related Sections Information

A descriptive analysis was done for the variable related data, and analysis results are summarized in this section.

### 4.9.1. Quality and software productivity

Below figure depicts the responses received under code quality category from the survey respondents.

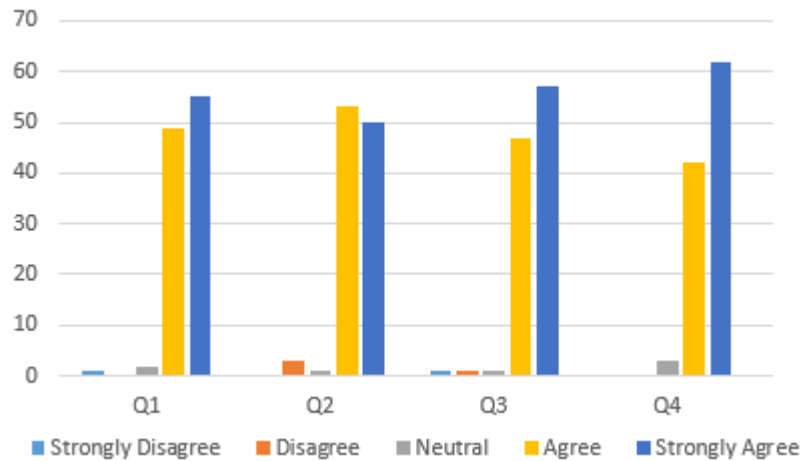


Figure 21 Quality and software productivity

Above per the above graph, nearly 55% of the respondents strongly agreed that having a high-quality code will eventually reduce the bug rate of the system, nearly 53% of the respondents agreed that every project should be equipped with automated code quality scanners such as sonar to continually monitors the coding standards and quality. Nearly 56 % of the respondents strongly agreed that every code commits should be evaluated for maintainability of the code before it gets merged with the production code. Nearly 61% of the despondence agreed that code review among peers should happen to improve the code quality of the product.

#### 4.9.2. Quantity and software productivity

Below figure depicts the responses received under the code quantity category from the survey respondents.

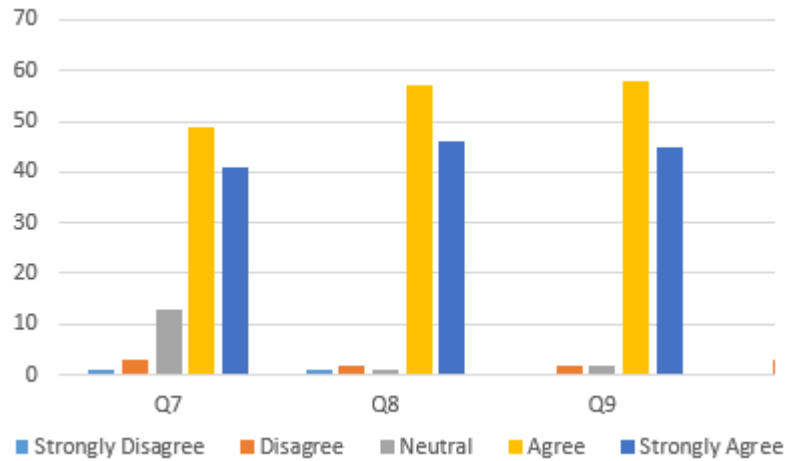


Figure 22 Quantity and software productivity

Above per the above graph, nearly 49 % of respondents believe that complex business problems usually requires and contains more LOC in the solution, nearly 55% of the respondents agreed that complex functionality usually contains more LOC in the associated function. Nearly 55% of respondents agreed that variable definition could be used to manipulate LOC, nearly 51% of respondents agreed that code quantity as an indicator of developer productivity.

#### 4.9.3. Code complexity for software productivity

Below figure depicts the responses received under code complexity category from the survey respondents.

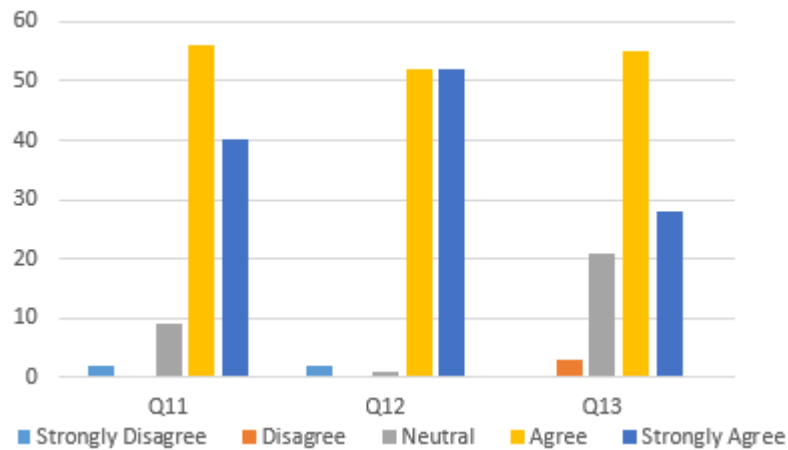


Figure 23 Code complexity for software productivity

Above per the above graph, nearly 55% of the respondents agreed that, when the complexity of the code gets higher then cyclometric complexity value will get increased, 52 % of the respondents agreed that when the complexity of the code gets higher then module design complexity value will get increased, Technically challenging and complex tasks such as developing an algorithm requires complex code structure as a solution.

#### 4.9.4. Work effort for software productivity

Below figure depicts the responses received under work effort category from the survey respondents.

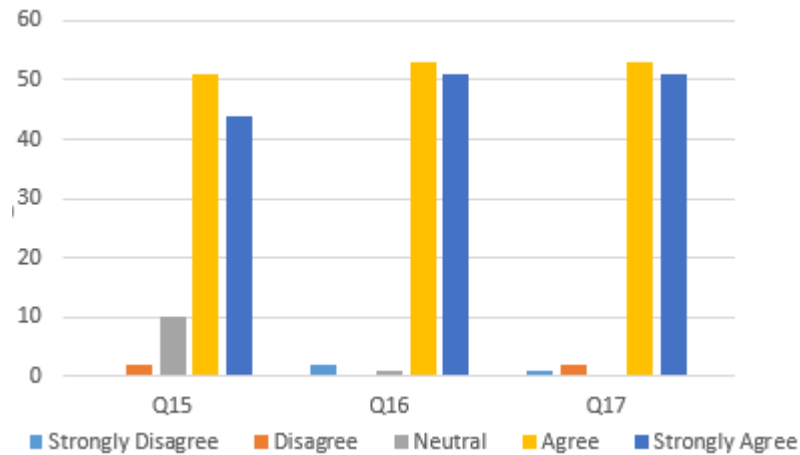


Figure 24 Work effort for software productivity

Above per the above graph, more than 50 % of the respondents agreed that proper planning regarding the solution before coding will helps to complete the tasks quicker than unplanned work. 53% of the respondents agreed that most experienced and talented software developers usually complete the task quicker than the average developers. Nearly 51% of the respondents agreed that accurate story point or work hours required estimation is a key factor in software development activities.

#### 4.10. Testing Hypothesis - Pearson’s Correlation Analysis

According to the obtained rating correlation analysis is used to identify the strength of the relationship between the variables. The following output is obtained using SPSS. To determine the connection between variables, correlation analysis was done. Standard averaging had been used for each variable to analyse the significance, by using Pearson Correlation Matrix. The correlation value “r” was defined as follows:

- .80 or higher - Very strong relationship
- .60 to .79 - Strong relationship
- .40 to .59 - Moderate relationship
- .20 to .39 - Weak relationship
- .00 to .19 – Very weak relationship

Significant value denotes the probability of correlation occurrence and a significant value less than 0.05 (5%) was considered significant.

#### 4.10.1. The correlation between code quality and productivity of a software developer

Below table presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.755 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

Table 5 The correlation between Code quality and productivity

		Quality	Productivity
Quality	Pearson Correlation	1	.755**
	Sig. (2-tailed)		.000
	Sum of Squares and Cross-products	107.000	80.732
	Covariance	1.000	.755
	N	108	108
Productivity	Pearson Correlation	.755**	1
	Sig. (2-tailed)	.000	
	Sum of Squares and Cross-products	80.732	107.000
	Covariance	.755	1.000
	N	108	108

\*\* . Correlation is significant at the 0.01 level (2-tailed).

It is indicating that code quality is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H1a is justified, and therefore the null hypothesis H1o is rejected.

#### 4.10.2. The correlation between code quantity and productivity of a software developer

Below table presents the two-tailed person correlation result for code quality vs the productivity of software developers. The value of the Pearson Correlation Coefficient at 0.555 which is significant at the 0.01 level indicates that there is a **moderately strong positive relationship** between the two variables.

Table 6 The correlation between Code quantity and productivity

		Quantity	Productivity
Quantity	Pearson Correlation	1	.555**
	Sig. (2-tailed)		.000
	Sum of Squares and Cross-products	107.000	59.419
	Covariance	1.000	.555
	N	108	108
Productivity	Pearson Correlation	.555**	1
	Sig. (2-tailed)	.000	
	Sum of Squares and Cross-products	59.419	107.000
	Covariance	.555	1.000
	N	108	108

\*\* . Correlation is significant at the 0.01 level (2-tailed).

It is indicating that code Quantity is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H2a is justified, and therefore the null hypothesis H2o is rejected.



#### 4.10.3. The correlation between code complexity and productivity of a software developer

Below table presents the two-tailed person correlation result for code complexity vs productivity of software developers. The value of the Pearson Correlation Coefficient at 0.693 which is significant at the 0.01 level indicates that there is a **strong positive relationship** between the two variables.

Table 7 Correlation between code complexity and productivity

		Correlations	
		Code Complexity	Productivity
Code Complexity	Pearson Correlation	1	.693**
	Sig. (2-tailed)		.000
	Sum of Squares and Cross-products	107.000	74.110
	Covariance	1.000	.693
	N	108	108
Productivity	Pearson Correlation	.693**	1
	Sig. (2-tailed)	.000	
	Sum of Squares and Cross-products	74.110	107.000
	Covariance	.693	1.000
	N	108	108

\*\* . Correlation is significant at the 0.01 level (2-tailed).

It is indicating that code Complexity is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H3a is justified, and therefore the null hypothesis H3o is rejected.

#### 4.10.4. The correlation between minimal work effort and productivity of a software developer

Below table presents the two-tailed person correlation result for minimal work effort vs productivity of software developers. The value of the Pearson Correlation Coefficient at 0.878 which is significant at the 0.01 level indicates that there is a **moderately strong positive relationship** between the two variables.

Table 8 Correlation between minimal work effort and productivity

		Minimal Work effort	Productivity
Minimal Work Effort	Pearson Correlation	1	.878**
	Sig. (2-tailed)		.000
	Sum of Squares and Cross-products	107.000	93.904
	Covariance	1.000	.878
	N	108	108
Productivity	Pearson Correlation	.878**	1
	Sig. (2-tailed)	.000	
	Sum of Squares and Cross-products	93.904	107.000
	Covariance	.878	1.000
	N	108	108

\*\* . Correlation is significant at the 0.01 level (2-tailed).

It is indicating that code quality is an important factor which should be considered to get an idea about the software developer's productivity. Based on the Pearson correlation value the alternative hypothesis H4a is justified, and therefore the null hypothesis H4o is rejected.

#### 4.11. Summary

This chapter reviewed the data analysis conducted and the insight created from the survey responses received from the software developers. Subsection discussed the

validity of the data collected, correlations between the identified factors and hypothesis validation to form the model, furthermore, this chapter also discussed the about the adjustment and fine-tuned incorporated in the module to get the more accurate result.

## 5. RECOMMENDATIONS AND CONCLUSION

### 5.1. Introduction

This chapter discusses the recommendation and conclusion based on the research findings and outputs. Section 5.2 details about the research conclusion after analysing the survey and interview outcomes.

Section 5.3 provides details about the limitation of this research. Finally, Section 5.4 describes the recommendation which can be taken to improve the software engineer's productivity in an organisation.

#### 5.1.1. Research conclusion one

Results of the correlation done by using the development activity data and the logarithm value correlation result and survey data correlation result indicate there is a positive correlation between productivity and Quantity, Quality and complexity. At the same time, there is a strong negative correlation between productivity and actual hours worked to complete the story.

#### 5.1.2. Research conclusion two

When comparing the correlation result of the development activity data, logarithm based values have higher correlation comparing to the non-logarithm based values, which provides powerful evidence that there is an existence of the non-linear relationship. Between productivity and Quantity, Quality, complexity and actual hours worked.

Table 9 Correlation Values

Relationship	Correlation value	The logarithm based correlation value
Productivity vs Code Quality	0.731	0.745
Productivity vs Code Quantity	0.714	0.716

Productivity vs Code Complexity	0.687	0.705
Productivity was actual work effort	-0.747	-0.752

### 5.1.3. Research conclusion three

As both the development activity data and survey data gathered from software developers are indicating the same result, hence, the model validity result is positive.

### 5.1.4. Research conclusion four

Based on the regression analysis done using logarithm based values, as the constant (intercept) was not significant and based on the regression coefficient, the following formula can be driven.

Developer's Productivity

$$= \frac{\text{Code Quantity}^{0.690} * \text{Code Quality}^{0.13} * \text{Code Complexity}^{0.17}}{\text{Actual Hours Worked}^{0.183}}$$

Considering the complexity of each factor represented in the equation, especially the code complexity and code Quantity, the equation should be used at a functional level within a class and then aggregated to the class level finally weighted average should be taken to the story.

## 5.2. Research Assumptions and Limitations

Code complexity value will be driven considering the class level because of the current limitations of available tools since it is assumed that each developer will be responsible for creating an entire class for the required feature.

Developers involvement in requirement gathering and understanding, creating the architectural and database related modification, cannot be considered to measure the productivity under the proposed model.

Developers involvement in requirement gathering and understanding, creating the architectural and database related modification, cannot be considered to measure the productivity under the proposed model.

### 5.3. Recommendation

To understand the individual productivity of a software developer following model can be used

Developer's Productivity

$$= \frac{\text{Code Quantity}^{0.690} * \text{Code Quality}^{0.13} * \text{Code Complexity}^{0.17}}{\text{Actual Hours Worked}^{0.183}}$$

Form the survey responses its was identified that the following actions could improve the quality of the work produced by the developer.

- Defining clear requirements in user stories.
- Defining and following a coding standard in the project
- Utilizing automated tools to scan for quality and maintainability of the code.
- Having regular code reviews.

Form the survey responses its was identified following actions could decrease the effort to complete a task

- Having a proper estimation before starting the development.
- Having proper planning before starting the development activities.
- Reduce the distractions such as meeting which won't add values etc.

### 5.4. Suggestion for Further Research

The research study was conducted only focusing on the productivity indicating factors which can be quantified; the model can be further validated and fine-tuned with various software development data from different software development methodologies.

Furthermore, there can be a more complexed model identified including other productivity indicating factors such as developer's initiatives, knowledge transfers, leadership etc.

## REFERENCES

Anselmo, D., & Ledgard, H. (2003). Measuring productivity in the software industry. *Communications of the ACM*, 46(11), 121-125. doi:10.1145/948383.948391

Lopez-Martin, C., Chavoya, A., & Meda-Campana, M. E. (2014). A machine learning technique for predicting the productivity of practitioners from individually developed software projects. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). doi:10.1109/snpd.2014.6888690.

Peck, C., & Callahan, D. (n.d.). A proposal for measuring software productivity in a working environment. *Proceedings of the Thirty-Fourth Southeastern Symposium on System Theory (Cat. No.02EX540)*. doi:10.1109/ssst.2002.1027063

Cusumano, M. A., & Kemerer, C. F. (1990). A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development. *Management Science*, 36(11), 1384-1406. doi:10.1287/mnsc.36.11.1384

Ondrej, M., Jiri, H., & Jan, H. (2012). Estimating Productivity of Software Development Using the Total Factor Productivity Approach. *International Journal of Engineering Business Management*, 4, 34. doi:10.5772/52797

Cedergren, S., & Larsson, S. (2014). Evaluating performance in the development of software-intensive products. *Information and Software Technology*, 56(5), 516-526. doi:10.1016/j.infsof.2013.11.006

Trendowicz, A., & Münch, J. (2009). Chapter 6 Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences. *Advances in Computers*, 185-241. doi:10.1016/s0065-2458(09)01206-6



Sudhakar, P., Farooq, A., & Patnaik, S. (2012). Measuring productivity of software development teams. *Serbian Journal of Management*, 7(1), 65-75. doi:10.5937/sjm1201065s

Balsamo, S., Marco, A. D., Inverardi, P., & Simeoni, M. (2004). Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5), 295-310. doi:10.1109/tse.2004.9

Carley, K. M. (2008). Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development. *SSRN Electronic Journal*. doi:10.2139/ssrn.2724745

Podjavo, I., & Berzisa, S. (2017). Performance Evaluation Of Software Development Project Team. Environment. Technology. Resources. *Proceedings of the International Scientific and Practical Conference*, 2, 118. doi:10.17770/etr2017vol2.2543

Sudhakar, G. P., Farooq, A., & Patnaik, S. (2011). Soft factors were affecting the performance of software development teams. *Team Performance Management: An International Journal*, 17(3/4), 187-205. doi:10.1108/13527591111143718

Hernández-López, A., Colomo-Palacios, R., García-Crespo, Á, & Cabezas-Isla, F. (2011). Software Engineering Productivity. *International Journal of Information Technology Project Management*, 2(1), 37-47. doi:10.4018/jitpm.2011010103

Flitman, A. (2003). Towards meaningful benchmarking of software development team productivity. *Benchmarking: An International Journal*, 10(4), 382-399. doi:10.1108/146357703104484999

Edberg, D. T., & Bowman, B. J. (1996). User-Developed Applications: An Empirical Study of Application Quality and Developer Productivity. *Journal of Management Information Systems*, 13(1), 167-185. doi:10.1080/07421222.1996.11518117

H.C. Shiva Prasad Damodar Suar, (2010), "Performance assessment of Indian software professionals", *Journal of Advances in Management Research*, Vol. 7 Iss 2 pp. 176 – 193

Amel Ben Hadj Salem Mhamdia , (2013), "Performance measurement practices in software ecosystem", *International Journal of Productivity and Performance Management*, Vol. 62 Iss 5 pp. 514 – 533

Ahmed, N. U., Ma, C. S., & Montagno, R. V. (1991). Measuring White-Collar Productivity. *American Journal of Business*, 6(1), 27-34. doi:10.1108/19355181199100005

Gustafsson, J. (2011). Model of Agile Software Measurement: A Case Study. Master of Science Thesis in the Programme Software engineering, Chalmers

Meyer, A. N., Zimmermann, T., & Fritz, T. (2017). Characterizing Software Developers by Perceptions of Productivity. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). doi:10.1109/esem.2017.17

Oberscheven, F. M. (2013). Software Quality Assessment in an Agile Environment. Faculty of Science of Radboud University in Nijmegen.

Germaine H. Saad, (2001), "Strategic performance evaluation: descriptive and prescriptive analysis", *Industrial Management & Data Systems*, Vol. 101 Iss 8 pp. 390 – 399

Agilemethodology.org. (2016). The Agile Movement. [online] Available at: <http://agilemethodology.org> [Accessed 13 Nov. 2016].

Scrum Alliance. (2016). Learn About Scrum. [online] Available at: <https://www.scrumalliance.org/why-scrum> [Accessed 12 Nov. 2016].

Graziotin, D. (2016). Towards a Theory of Affect and Software Developers' Performance. arXiv preprint arXiv:1601.05330.

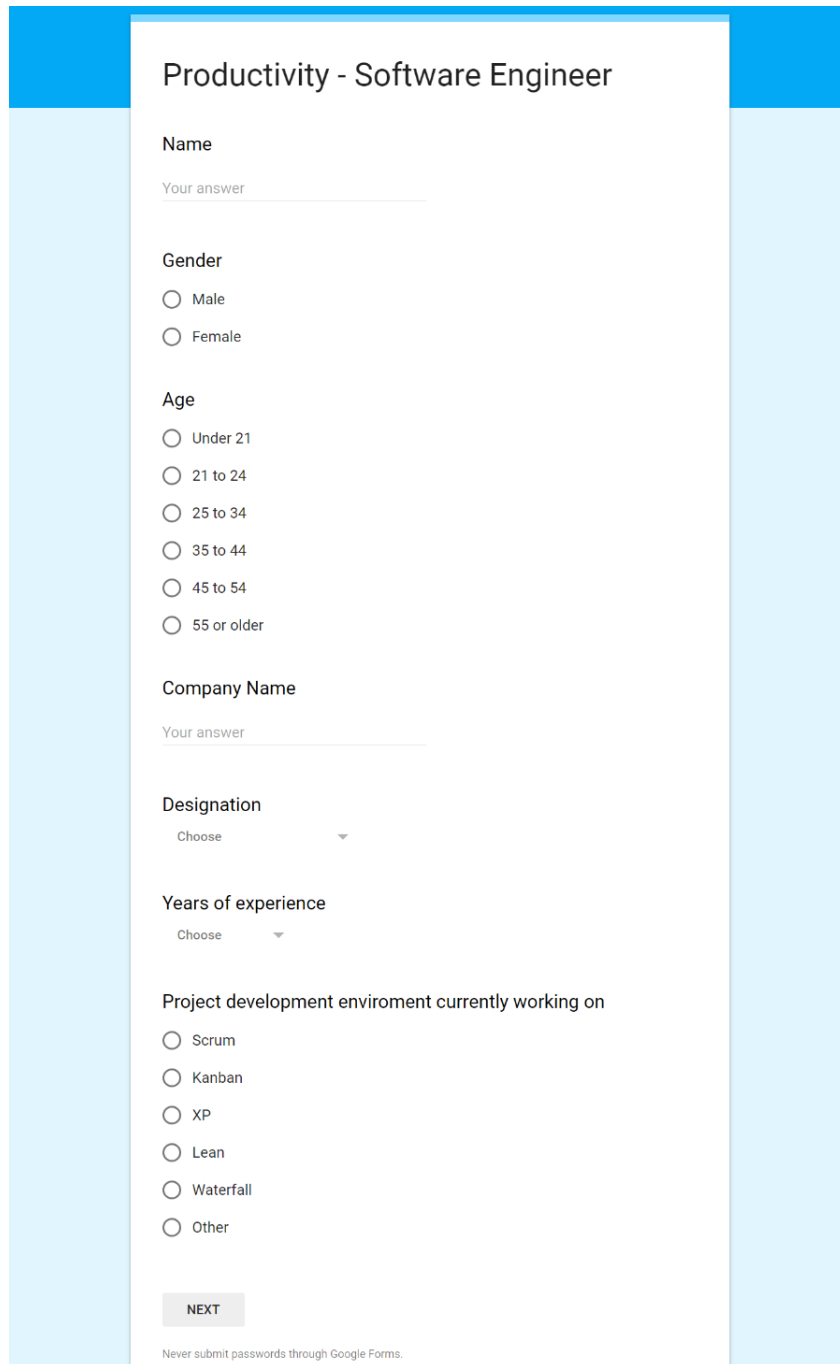
Baggelaar, H., & Klint, P. (2008). Evaluating Programmer Performance. Amsterdam: sn.

Peck, C., & Callahan, D. (n.d.). A proposal for measuring software productivity in a working environment. Proceedings of the Thirty-Fourth Southeastern Symposium on System Theory (Cat. No.02EX540). doi:10.1109/ssst.2002.1027063

## APPENDIX A: TITLE

Distributed questionnaire can be found under following URL :

<https://docs.google.com/forms/d/e/1FAIpQLSfEAO1huwHbZ0L2OQoY7nydJc7pKcUbgmFIV8Tm4u62Vq5Ow/viewform>



The image shows a Google Form titled "Productivity - Software Engineer". The form is displayed on a white background with blue accents at the top and bottom. The questions are as follows:

- Name**: A text input field with the placeholder "Your answer".
- Gender**: A radio button selection with options "Male" and "Female".
- Age**: A radio button selection with options "Under 21", "21 to 24", "25 to 34", "35 to 44", "45 to 54", and "55 or older".
- Company Name**: A text input field with the placeholder "Your answer".
- Designation**: A dropdown menu with the placeholder "Choose".
- Years of experience**: A dropdown menu with the placeholder "Choose".
- Project development enviroment currently working on**: A radio button selection with options "Scrum", "Kanban", "XP", "Lean", "Waterfall", and "Other".

At the bottom of the form, there is a "NEXT" button and a small disclaimer: "Never submit passwords through Google Forms."

Figure 25 Survey Questions part 1

## Productivity - Software Engineer

### Quality

A practical and measurable standard on code quality will help towards improving the quality of a software system.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Code review among peers helps to improve the quality of the software system.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Having a pre-defined coding standard is a best way to guide developers towards a proper structured quality code, which can later be easily understood and easily modified by another developer.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Disagree

Having a quality standard as a benchmark and measuring each code commit against it will improve the system performance as well as the security aspect of the system.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Implementing a code quality monitoring system will radically improve the code quality of your team's products.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

I believe code quality is an indication of productivity.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

[BACK](#) [NEXT](#)

Never submit passwords through Google Forms.

This form was created inside Department of Computer Science & Engineering - UOM. Report Abuse - Terms of Service - Additional Terms

Google Forms

Figure 26 Survey Questions part 2

# Productivity - Software Engineer

## Quantity

Lines of code (LOC) can be considerable as a Quantity measurement matrix

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

While measuring lines of code, padding the code with unnecessary lines should be identified and removed from the line's count.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Avoiding variable declaration from lines of code will help to eliminate code Quantity manipulation.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

I believe code quantity is an indication of productivity.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

BACK

NEXT

Never submit passwords through Google Forms.

This form was created inside Department of Computer Science & Engineering - UOM. Report Abuse - Terms of Service - Additional Terms

Google Forms

Figure 27 Survey Questions part 3

# Productivity - Software Engineer

## Code Complexity

Code complexity has a high impact on developer productivity.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Code complexity can be measure various matrixes such as Cyclomatic Complexity, Module Design Complexity.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

A better code complexity measurement can be achieved by combining different complexity measurement model together.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

I believe code complexity is an indication of productivity.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

BACK

NEXT

Never submit passwords through Google Forms.

This form was created inside Department of Computer Science & Engineering - UOM. Report Abuse - Terms of Service - Additional Terms

Google Forms

Figure 28 Survey Questions part 4

# Productivity - Software Engineer

## Actual Work Effort

Measurement about total hours taken to complete a task is an important factor when it comes to software development productivity measurement.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

I have improved myself by monitoring the actual hours taken to complete my previous tasks.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Every development story should be properly analyzed for work effort required and time boxed before starting the actual development work.

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

I believe work effort in an indication of software developers productivity I believe

	1	2	3	4	5	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

BACK

SUBMIT

Never submit passwords through Google Forms.

This form was created inside Department of Computer Science & Engineering - UOM. Report Abuse - Terms of Service - Additional Terms

Google Forms

Figure 29 Survey Questions part 5