Chapter 1: Introduction

1.1 General Concepts

What Is Agile?

Agile methodology is a substitute to traditional project management, typically used in software development. It helps a team to respond for unpredictability through incremental, iterative work, known as sprints. Agile methodologies are an alternative to waterfall, or traditional sequential development [1].

Why Agile?

Agile development methodology provides a way to assess the direction of a project throughout the development lifecycle. This is achieved through regular iteration of work, known as sprints, at the end of which teams must present a potentially shippable product increment. By focusing on the repetition of abbreviated work cycles as well as the functional product they yield, agile methodology is described as “iterative” and “incremental” [1].

In waterfall model, development teams only have one chance to get each aspect of a project right. In an agile modeling, every aspect of development — like requirements, design, etc. — is continually revisited throughout the lifecycle. When a team stops and re-evaluates the direction of a project every two weeks, there’s always time to steer it in another direction.

Agile Model:

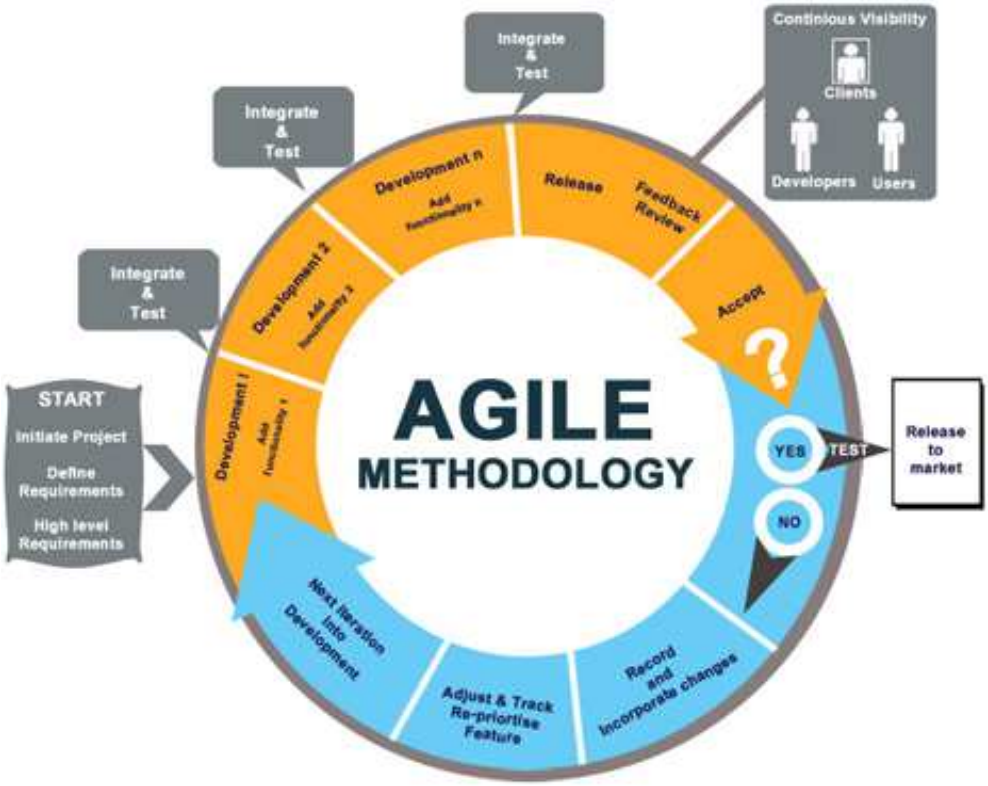

Figure 1: Agile Model [1]

Budgeting

- It defines how much we have to spend based on the scope of the work
- Also, it tends to ignore the cone of uncertainty

Estimation

- Presents an approximation of effort and duration based on size and project nature
- Focused by the cone of uncertainty (a range based on knowledge)

Planning

- Defines tasks and allocates resources
- Focused on the narrow part of the cone of uncertainty (a much smaller range)

1.2 Motivation

At early stages of software development, effort must be estimated to come up with a planned schedule and budget. Software processes constantly evolve as new and different technologies and applications are developed and used. Especially, in current software industry, where changes are arbitrary, an evolving system is required for the problem of effort estimation especially in agile environment.

The latest Release of **ISBSG (International Software Benchmarking Standard Group)** volume 11 [16] has reported the fact that only 6.4 % of the total projects (the one which are included in survey) in the world are developed through agile methodologies in spite of the several advantages of the methodology like quick delivery, high customer involvement, iterative and incremental model, always welcomes requirement changes etc. The graph below depicts the use of different methodologies to develop software's.

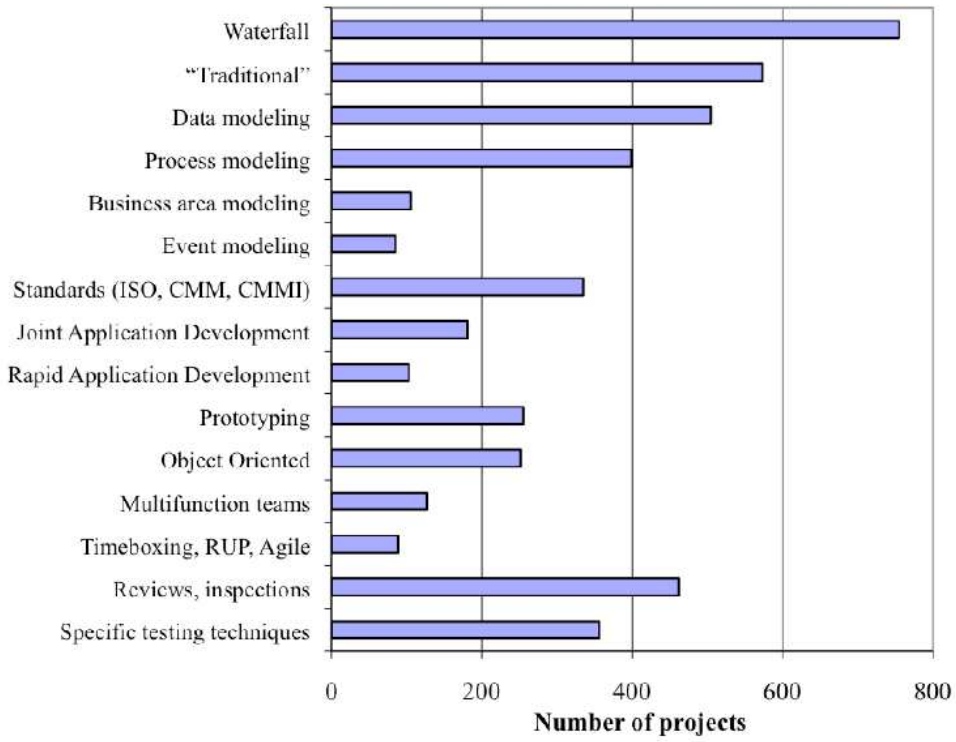

Figure 2: Comparision of Modelling techniques [16]

Development techniques	Projects	Percent
Waterfall	745	
"Traditional"	713	
Specific techniques described	1384	
Data modeling	505	36.5 %
Process modeling	399	28.8 %
Business area modeling	106	7.7 %
Event modeling	85	6.1 %
Standards (ISO 9000; CMM, CMMI)	335	24.2 %
Joint Application Development	181	13.1 %
Rapid Application Development	103	7.4 %
Prototyping	255	18.4 %
Object Oriented Analysis/Design, UML	252	18.2 %
Multifunction teams	127	9.2 %
Timeboxing, RUP, Agile	89	6.4 %
Reviews, inspections, walkthroughs	462	33.4 %
Specific testing techniques	356	25.7 %

Table 1: Percentage usage of Different Development technique [16]

Estimation this day's has been a lightning rod for the discussion in all methods (agile, waterfall, iterative or water fountain) with the issues of predictability and standardization. Because of the controversy this is an area where a wide range of hybridization has always occurred. Organizations adjust techniques which can be fitted over governance structures, culture and risk profiles. There is no one size fits for all solution.

When surveyed with the recent research [5, 12, 22, 26] that is ongoing in agile methodology it has been found that one of the reason for not using agile methodologies is lack of efficient estimation methodology which can accurately estimate the effort and time required to develop a project. So in order to support or to enhance the use of agile methodologies need for some good estimation techniques arises which can provide accurate results.

1.3 Related work

In software development organizations, the main aim is to start a project and finish it within acceptable schedule and budget. The main drivers affecting the budget are the number of employees and time. These two come together to form a measure called effort. Effort is the most important factor in a software project determining the cost. Estimating effort is a very important factor affecting the quality of software.

There has not been any significant research in the field of estimation of effort in agile environments using machine-learning methods. A recent research paper from this domain [5], reports successful use of user stories for effort prediction. Such approach is well suited for agile software project where requirements are developed along with the project. But the performance of the tool for keyword extraction could depend on the language so this can be a threat to its validity.

Evita Coelho and Anirban Basu [27] have presented their work in agile estimation by creating a model similar like function point, they have modeled story points and on the basis of story points they have predicted the required effort but the results of their research is not up to the mark.

Another recent research from Thomas Cagley VP, David Consulting Group [28] provides a path for incorporating the use of function points into agile estimation techniques. The research shows that agile Estimation Using Functional Metrics has presented a path for integrating the discipline found in functional metrics with the collaborative approaches found in agile estimation.

There is a widely used method for agile estimation i.e. planning poker [6] which is heavily used to estimate the cost and duration of a project. This method is used to derive the estimates on the basis of expert opinion and historical data. However, this method is not useful in case of

unavailability of both the above factors. Further, this method may generate the different estimates for same project depending on the intuition of the estimators. Hence inaccurate estimation can lead to heavy losses which nobody wants.

Our research shows that one can use project related factors to aid the estimation process. This notion was supported by a recent research [12], which found out a few key project factors that have an effect on agile development and development effort. Out of several potential factors, the following factors were found to have considerable effect on the development process

Key factors influencing agile development

1. Development Platform
2. Function Point
3. Max. Team Size
4. Architecture
5. Operating System
6. Development Tool
7. Development Techniques
8. CMMI
9. ISO

Apart from estimation in agile environment there are various researches going for estimation in traditional methodologies as well. These researchers are driven by Artificial Neural Network which helps one to create an automated tool for effort predication. A recent research by C.W. Dawson [15] states the use of Neural network for software project estimation also shown great results in terms of mean square error.

Another similar concept given by A. B. Nassif, L. F. Capretz and D. Ho [17] they have built a model to predict software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model.

In another study [13], Bakeles and Turhan used various machine learning methods for effort estimation. They were able to obtain better results than algorithmic estimation methods.

1.4 Problem Statement

From the forgoing section we can conclude that if one can identify the major project characteristics factor which can affect the effort required, then it would be very helpful to improve the accuracy of effort estimation techniques.

It was also concluded that software effort estimation must be handled using an evolving system like artificial neural network rather than a static one. This study supported our notion of using machine learning methods for estimation. **.** This technique will help one to achieve lower Mean relative error (MRE) value and will help to reduce the losses due to inaccurate estimation. Hence problem statement of this thesis is as follows:

This thesis aims for developing an effort estimation technique by identifying intense project characteristic based on artificial neural network.

The main reason for using ANN for this problem is to keep the estimation process up-to-date by incorporating up-to-date project data. If the effort estimation process evolves with new projects, the estimation model is kept up-to-date. This overcomes the main problem in simple estimation models.

1.5 Scope of work

Thesis proposes an automated estimation approach, which can be an improvement over the most commonly used estimation method in agile environments discussed above. Proposed methodology is an artificial neural network based effort estimation method for agile software

projects, which takes input as function point, Team size and weights of different project factors and produces an estimated effort required to develop a project.

The scope of the work is not just confined to predict the effort required for agile software project, it also identify the intensity levels of the various project factors like complexity, Team etc which can affect the effort. This Intensity level will help one to find the weight of different metrics shown in appendix 1, which indeed we tell one about the dominance among other metrics.

Hence Scope of the work can be stated as follows:

- The general idea is to identify the intensity of various project related factors which are classified in 10 metrics namely complexity, team, performance etc. The value of these metrics will affect the effort required in the project, the values of the factors are from the set {very low, low, medium, high, very high} and these values will be useful to calculate the weight. A Cumulative weight of all the 10 metrics is found and with the help of function point analysis the functions points of the project are calculated.

- Now with the help of a trained artificial neural network, whose inputs will be Cumulative weight of different metrics, team size and function points for the project, will get a value of effort as the output of the ANN.

- The work is been validated with the help of the famous Chinese data set which contains about 100 projects with their actual time taken and actual effort taken to develop an software through agile methodologies. The dataset is shown in appendix-2.

1.6 Thesis organization

Chapter 1 begins with General introduction and related work. It addresses the topics like Motivation, Problem Statement, Scope, Related work and thesis organization.

Chapter 2 provides a detailed description about agile methodologies, explains different types of agile methodologies, Effort estimation and Benefits of Accurate estimation.

Chapter 3 shows different types of popular software estimation techniques which were or which are used in today's world. It includes FPA, COCOMO 2, Planning poker, estimation using ANN.

Chapter 4 presents the proposed research methodology which checks the flexibility of using ANN in Agile for effort estimation.

Chapter 5 shows the implementation of the proposed methodology also the tools used in it.

Chapter 6 presents the results and analysis part of the proposed methodology.

Chapter 7 concludes the thesis.

Chapter 2: Agile Methodologies and Effort Estimation

2.1 Introduction to Agile Methodologies

Agile Software Development Methods are nowadays widely used. The procedure to carry out such kind of projects takes especially constructive aspects and short realization times into account. From the Software Measurement point-of-view not every metrics and methods from conventional lifecycle models can be used without changes. Within this thesis we wanted to investigate especially the aspect of effort estimation activities for agile software development projects. For this we want to show the possibilities and borders of effort estimation tasks and the new challenges [1].

The following questions are examined:

- Which conditions do projects that are executed with help of agile process models assume?
- Which basic approaches (state of the art) of the effort estimation can be identified for agile executed projects at present?
- Are the classic methods of the estimation procedures, like Function Points or COCOMO applicable?
- Which experiences can be found in the industrial and academic surroundings with the use of agile procedures?

Beside the clarification of these questions, a short overview is given to the key characteristics of agile software development methods. For this we want to consider various agile methodologies like Scrum, Agile Modeling, Extreme programming etc, which are explained in the Coming session.

2.2 Types of Agile methodologies

Extreme Programming

Extreme Programming (XP) is a usual and wide spread agile software development method. Therefore, we have used XP as a reference model for agile software development.
In the center of software development projects should be the human and not documents, processes and tools. XP provides a set of practices, values and principles [29]:

- Values (e.g. communication, simplicity, feedback, courage)
- Principles (e.g. incremental changes, honest measuring).
- Practices (e.g. pair programming, short version cycles)

These set is derived from best practices and should help one to carry out successful software development projects. To realize these totally different project characteristics, the agile software development has established a different type of product life cycle in comparison with traditional life cycle models like waterfall-model [29].

Below Figure 3 shows the necessary process steps of a XP-project. The major element of the XP life cycle is the “Iteration”. The iteration is a recurring event in which an actual version is edited for example by:

- Adding additional functionality.
- Correcting errors.
- Removing unnecessary functionality.

Each software version will be validated through an acceptance test. In XP the duration of each iteration is very short (1 until 4 weeks) in comparison with traditional software development.

General Way of Developing Software through XP

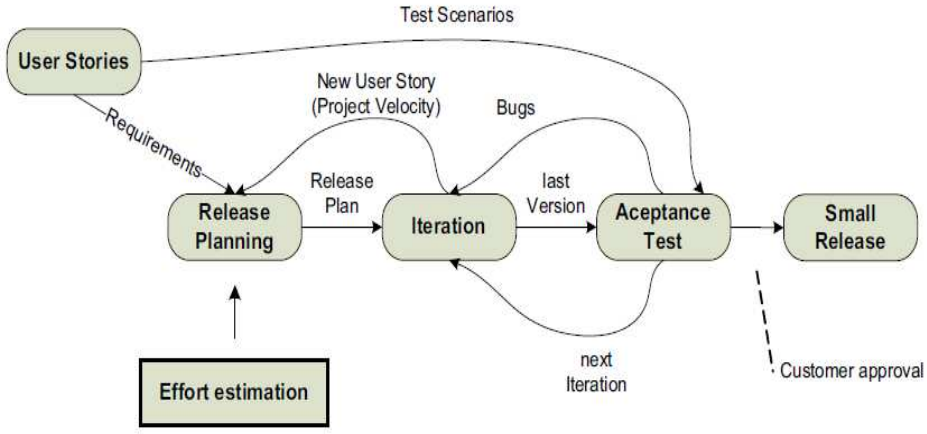

Figure 3: General Way of Developing Software through XP [29]

Agile Modeling (AM)

Modeling is an important step in software development. It enables software developers to get a blue print about complex issues before addressing them in programming. Agile Modeling (AM) was established by Scott Ambler in 2002. It is a collective set of values, principles, and practices for modeling software that can be used for software development project in an effective and easy manner [1].

The values of AM, which are considered to be an extension to the values of XP include: communication, simplicity, feedback, courage, and humility. Humility means to admit that you may not know everything; others may know things that you do not know, and thus, they may provide useful contribution to the project [1].

Again, the principles of AM are quite similar to those of XP, such as assuming simplicity, embracing changes, incremental change of the system, and rapid feedback. In addition to these

Principles, AM principles include the knowledge of the purpose for modeling; having multiple effective models; the content is more important than the representation; keeping open and honest communication between parties involved in the development process; and finally, to focus on the quality of the work [1].

The practices of AM have some commonalities with those of XP, too. An agile modeler needs to follow these practices to create a successful model for the system. AM practices highlight on active stakeholder participation; focus on group work to create the suitable models; apply the appropriate artifact as UML diagrams; verify the correctness of the model, implement it and show the resulting interface to the user; model in small increments; create several models in parallel; apply modeling standards; and other practices [1].

Agile Model Driven Development (AMDD) is the agile version of model driven development. To apply AMDD, an overall high level model for the whole system is created at the early stage of the project. During the development iterations, the modeling is performed as planned per iteration. Usually, AM is applied along with other methodologies, such as Test Driven Development (TDD), and Extreme Programming (XP), to get the best results [1].

AM basically creates a mediator between rigid methodologies and lightweight methodologies, by suggesting that developers communicate architectures through applying its practices to the modeling process [2]. In a nut, agile modeling defines a collection of values, principles, and practices which describe how to streamline the modeling and documentation efforts. It is usually applied in conjunction with agile implementation techniques for good results.

SCRUM

SCRUM methodology was initiated by Ken Swaber in 1995. It was practiced before the announcement of Agile Manifesto. Later, it was included into agile methodology since it has the same underlying concepts and rules of agile development. SCRUM has been used with the objective of simplifying project control through simple processes, easy to update documentation and higher team iteration over exhaustive documentation [4].

SCRUM shares the basic concepts and practices with the other agile methodologies, but it comprises project management as part of its practices. These practices guide the development team to find out the tasks at each development iteration. In addition to the practices defined for agility, one main mechanism recommended by SCRUM is to build a backlog. A backlog is a place where one can see all requirements pending for a project, sized based on complexity, days or some other unit of measure the team decides. Inside a product backlog, there is a simple sentence for each requirement; something that will be used by the team to start discussions and putting details of what is needed to be implemented by the team for that requirement [4].

For SCRUM, three main roles are defined as shown in Fig 4. The first role is the product owner, who mainly would be the voice of business. The second role is the SCRUM team which comprises developers, testers, and other roles. SCRUM master, the third role, is responsible for keeping the team focused on the specific tasks [4][3].

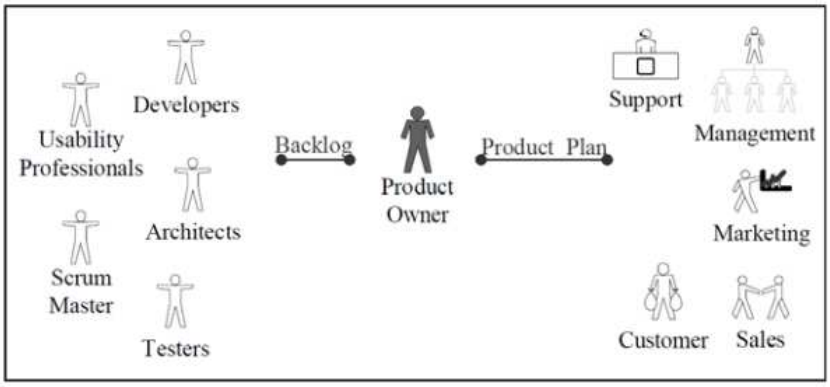

Figure 4: Key roles and interaction artifacts in SCRUM [3]

The process of development using SCRUM divides the project into phases. In each phase, one feature is fully developed, tested, and become ready to go to production. The team does not move to a new phase until the current phase is completed. Whether what is being done adds value to the process or not, is the main concern of each phase.

Current studies on traditional SCRUM development have shown that despites its advantages, it is not best suited for products where the focus is on usability [3]. It fails to address usability needs of the user, because product owners keep their focus mainly on business issues and forget about usability. Since product owners usually come from business background, they lack the experience, skills, and motivation to design for user experiences. Moreover, traditional agile methodologies are not concerned about the user experience vision, which drives the architecture and is essential for ensuring a coherent set of user experiences.

Briefly, SCRUM is considered an iterative, incremental methodology of software development. It was proposed for software development projects, and at the same time, it can be used as a program management approach.

2.3 Characteristics of Agile methodologies

Many technological ambitious products were designed with new complex functionality. The demand for functions establishes a need for new software requirements to deliver new functionality. Due to the fast alteration and the high cost of change in the late life cycle phases the agile software development method becomes more important in this field of application [23][25].

Agile software development methods like Extreme Programming try to decrease the cost of change and therewith reduce the overall development costs. Agile methods try to avoid the deficits of classic software development procedures.

Mostly the following methodologies are considered to reach this aim:

- short release cycles,
- simple design,
- continuous testing,
- refactoring,
- collective ownership,
- coding standard and
- Continuous integration.

2.4 Effort Estimation

Software development effort estimation is the process of predicting the most efficient use of effort required to develop or to maintain software based on incomplete, uncertain and/or noisy input. Effort estimates may be used as input to project plans, iteration plans, budgets, and contractual work.

Estimation and planning are often confused with each other. Although they are related topics, but estimation is not planning, and planning is not estimation. Estimation should be treated as an unbiased, analytical process whereas planning should be treated as a biased, goal-seeking process. With estimation, it is hazardous to want the estimate to come out to any particular answer. The goal is accuracy; the goal is not to seek a particular result. However, the goal of planning is to seek a particular result. We intentionally and aptly bias our plans to achieve specific results. We plan specific processes to reach a specific outcome.

Estimates form the foundation for the plans, but the plans do not have to be the same as the estimates. If the estimates are dramatically different from the targets, the project plans will need to recognize that gap and account for a high level of risk. If the estimates are close to the targets, then the plans can assume less risk.

Both estimation and planning are important, but the fundamental differences between these activities mean that mixing the two tends to lead to poor estimates and poor plans.

2.5 Classification of Effort Estimation

There are many different estimation methods, which are classified into three major categories:

- **Formal/Algorithmic estimation**

  These models (ex. COCOMO, Function Point analysis) rely on the experience gained on previous software projects in the sense that they connect size and effort value by means of one of the explicit function forms, by applying regression analysis method. In doing so, most widely used are linear and exponential dependence.

- **Expert estimation**

  These methods (ex. Wideband Delphi, Planning Poker) are based on consultation of one or more people considered as experts in software development. The estimate is produced based on judgmental processes.

- **Analog estimation**

  It is characteristic for these models that in order for estimations to be made, analogies are used between the new project and some of the already completed ones. Comparisons are made between the suggested project and similar projects for which data in respect of cost, time and effort are known. These models (ex. ESTOR, ANGEL) require as much data as possible concerning implemented projects.

2.6 Benefits of Accurate Estimation

- **Improved status visibility for the stakeholders**

  One of the best ways to track progress is to compare planned progress with actual progress. In case of real planned progress (that is, based on accurate estimates), it's possible to track progress by sticking to the plan. Good estimates thus provide vital support for project tracking.

- **Higher quality**

  Accurate estimates help avoid schedule-stress-related quality problems. About 40% of all software errors have been found to be caused by stress; those errors could have been avoided by scheduling appropriately and by placing less stress on the developers. When schedule pressure is extreme, about four times as many defects are reported in the released software as are reported for software developed under less extreme stress.

- **Better coordination with non-software functions**

  Software projects usually need to coordinate with other business functions, including testing, document writing, marketing campaigns, sales staff training, and software support training. If the software development schedule is not correct, then it can cause related functions to fall, which can cause the entire project schedule to fall. Better software estimates allow for better coordination of the whole project, including both software and non-software activities.

- **Better budgeting**

  Although it is almost too obvious to state, accurate estimates support accurate budgets. An organization that doesn't support accurate estimates undermines its ability to forecast the costs of its projects.

- **Increased credibility for the development team**

  It is a common practice in software development that after a project team creates an estimate, the managers take the estimate and turn it into an business target. A project team that holds its ground and insists on a realistic development plan based on accurate estimates will improve its credibility within its organization.

- **Early risk information**

  One of the most common wasted opportunities in software development is the failure to correctly interpret the meaning of an initial mismatch between project goals and project estimates.

Chapter 3: Software Estimation Techniques

There are many models for software estimation available and running in the industry. Researchers have been working on various estimation techniques since 1965. Initial work in estimation was based on regression analysis or mathematical models of other domains. Among many estimation models expert estimation, COCOMO, Function Point and derivatives of function point like Use Case Point, Object Points are most commonly used. While Lines of Code (LOC) is most commonly used as a size measure. IFPUG FPA originally invented by Allen Alrecht at IBM has been adopted by most in the industry as alternative to LOC for sizing development and enhancement of business applications. Function Point Analysis provides measure of functionality based on end user view of application software functionality. Some of the commonly used estimation techniques are as follows:

3.1 Lines of Code (LOC)

It is a formal method to measure size by counting number of lines of code. LOC is typically used to get the amount of effort that will be required to develop a program, also to estimate programming productivity or maintainability once the software is produced. Lines of Code (LOC) have two variants- Physical LOC and Logical LOC. While two measures can vary significantly.

3.2 Function Point

Software cost estimation is the process of predicting the effort for developing software. Function points are a metric since it provides a sizing gauge for products very early in the development cycle. Function Points represent the effort put into developing the desired features [28]. Once the functions of the product are identified, they are categorized into

distinct types. They are then assessed for their complexity, and function points are assigned to the features. In this paper, function points are used as base cost estimation metric at agile software projects. Common agile projects use story points for estimating the work. Desired features are identified and the total number of story points is estimated at the beginning phase of the project. The total number of story points is reduced by the completion of each user story while the project is progressing. As the story points are measured via comparisons with other stories, the total number of story point can fluctuate with small variations of the base story point.

On the other hand, function points are absolute values. The function points quantify the size and complexity of an application based on that application's inputs, outputs, inquiries, internal files, and interfaces. They are measured based on the complexity of the desired features and the interface of the user story itself. Thus, the total number of function points is more stable than any individual story point.

3.3 COCOMO 81

COCOMO 81 (Constructive Cost Model) is an empirical estimation scheme. [7] It is used for estimating effort, cost, and schedule for software projects. It was derived from the large data sets [19] from 63 software projects ranging in size from 2,000 to 100,000 lines of code. These data were analyzed to discover a set of formulae that were the best fit to the observations. These formulae link the size of the system and Effort Multipliers (EM) to give the effort required to develop a software system.

In COCOMO 81, effort is expressed as Person Months (PM) and it can be calculated as

$$PM = a * Size^{b} * \prod_{i=1}^{15} EM_{i}$$

Where “a” and “b” are the domain constants in the model. It contains 15 effort multipliers. This estimation scheme accounts the experience and data of the past projects, which is extremely

complex to understand. Cost drives have a rating level that shows the impact of the driver on development effort. These rating can range from Extra Low to Extra High. For the purpose of analysis, each rating level of each cost driver has a weight associated with it. The weight is called Effort Multiplier(EM). The average EM assigned to a cost driver is 1.0 and the rating level associated with that weight is called Nominal [7].

3.4 COCOMO II

It is an enhanced scheme for estimating the effort for software development activities, which is called as COCOMO II. In COCOMO II, the effort requirement can be calculated as:

$$PM = a * Size^{E} * \prod_{i=1}^{17} EM_i$$

$$E = B + 0.01 * \sum_{i=1}^{5} SF_j$$

COCOMO II is associated with 31 factors; LOC measure as the estimation variable, 17 cost drives, 5 scale factors, 3 adaptation percentage of modification, 3 adaptation cost drives and requirements & volatility. Cost drives are used to capture characteristics of the software development that affect the effort to complete the project. COCOMO II used 31 parameters to predict effort and time [11] [12] and this larger number of parameters resulted in having strong co-linearity and highly variable prediction accuracy. Besides these meritorious claims, COCOMO II estimation schemes are having some disadvantages. The underlying concepts and ideas are not publicly defined and the model has been provided as a black box to the users [26]. This model uses LOC (Lines of Code) as one of the estimation variables, whereas Fenton et. al [27] explored the shortfalls of the LOC measure as an estimation variable. The COCOMO also uses FP (Function Point) as one of the estimation variables, which is highly dependent on development the uncertainty at the input level of the COCOMO yields uncertainty at the output, which leads to gross estimation error in the effort estimation [33]. Irrespective of these drawbacks,

COCOMO II models are still influencing in the effort estimation activities due to their better accuracy compared to other estimation schemes.

3.5 Planning Poker

Planning poker is the mostly used technique for estimation in agile environment. This technique is highly depended on the expert who takes part in estimation process. Participants in planning poker include all of the developers on the team [6]. The term "developers" refers to all programmers, testers, analysts etc. On an agile project, this will typically not exceed ten people. At the start of planning poker, each estimator is given a deck of cards. Each card has written on it one of the valid estimates. Each estimator may, for example, be given a deck of cards that reads number from Fibonacci Series 0, 1, 2, 3, 5, 8, 13 and 21. Fibonacci numbers have been found to be a very useful estimation sequence because the gaps in the sequence become appropriately larger as the numbers increase. These non-linear sequences work well because they reflect the greater uncertainty associated with estimates for larger units of work i.e. in case of 13 story points, it is difficult to argue whether the card is worth 13 points or 12 points [6].

For each user story to be estimated, a moderator reads the description. The moderator is usually the product owner (customer) or an analyst. The product owner answers any questions that the estimators have. The goal in planning poker is not to derive an estimate that will withstand all future scrutiny. Rather, the goal is to be somewhere well on the left of the effort line, where a valuable estimate can be arrived at cheaply. After all questions are answered, each estimator privately selects a card representing his or her estimate. Cards are not shown until each estimator has made a selection. At that time, all cards are simultaneously turned over and shown so that all participants can see each estimate.

It is very likely at this point that the estimates will differ significantly. If estimates differ, the high and low estimators explain their estimates. The moderator can take any notes he thinks

will be helpful when this story is being programmed and tested. After the discussion, each estimator re-estimates by selecting a card. In many cases, the estimates will already converge by the second round. But if they do not, the process is repeated. The goal is for the estimators to converge on a single estimate that can be used for the story.

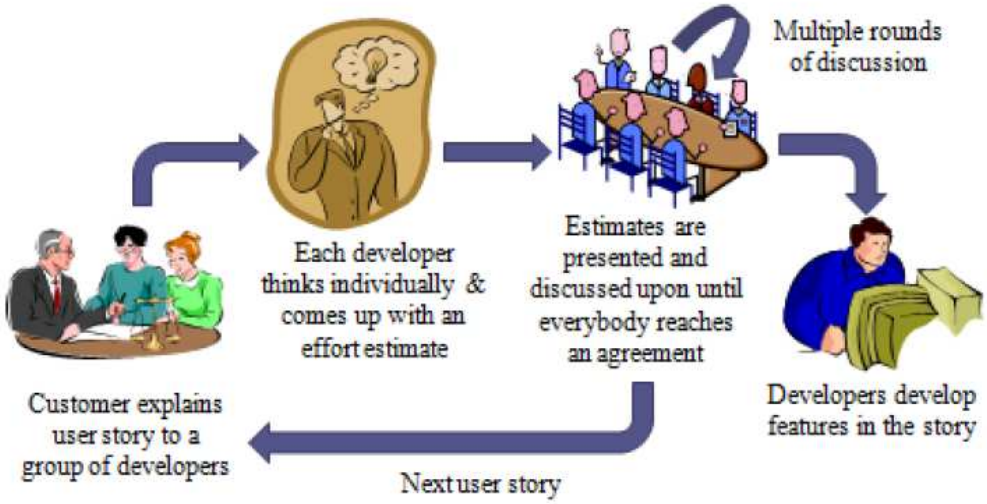

Figure 5: Planning Poker [6]

Shortcomings of Planning Poker

Based on a research [6], it has been concluded that the following areas need to be addressed:

- **High Magnitude of Relative Error (MRE) in estimation**

  Magnitude of Relative Error is a widely used measure for evaluating the estimation accuracy of different models. For a single estimate, it is defined as:

$$MRE = \frac{|\text{Actual Effort} - \text{Estimated Effort}|}{|\text{Actual Effort}|}$$

  Mean MRE is used to quantify the accuracy for the complete model. Based on the estimation data collected from the enterprise for planning poker [6], this value comes out to be 1.0681 or 106.81% which is very high and can be reduced.

- **Strong over-confidence in accuracy of estimates**

  Software development projects frequently have over-optimistic effort estimates and over-confident assessments of estimation accuracy. It has been observed that [6] there is large percentage of projects which are either over or under estimated using the Planning poker:

Accurate Estimates	30
Over-estimates	49
Under-estimates	44

Table 2: Estimates using Planning Poker [6]

- **An expert-dependent method**

  Even though planning poker takes every developer's estimate into consideration, the bias towards estimates from experts cannot be fully avoided. From the perspective of managers and developers at the enterprise, an expert is more likely to convince his/her opinion to the rest of the team than a novice developer in the team. And it has also been observed that in absence of an expert in the team, the accuracy of estimates decreases substantially.

3.6 Estimation using artificial neural network

Artificial Neural Network (ANN) is a network composed of artificial neurons or nodes which emulate the biological neurons [11]. ANN can be trained to be used to approximate a non-linear function, to map an input to an output or to classify outputs. Now a day's ANN is highly used for estimating the effort required for developing software's by traditional methodologies [15] [17].

The most prominent topology of ANN is the feed-forward networks. Feed forward ANN layers are usually represented as input, hidden and output layer. If the hidden layer does not exist, then this is called perceptron. The perceptron can map an input to an output if the relationship between is linear. If the relationship between the input and output is non-linear, one or more

hidden layers will exist between the input and output to accommodate the non-linear properties.

Several types of feed-forward NN with hidden layers exist. These include Multilayer Perceptron (MLP), Radial Basis Function neural Network (RBFNN) and General Regression (GRNN). A MLP contains at least one hidden layer and each input vector is represented by a neuron. The number of hidden neurons varies and can be determined by the trial and error so that the error is minimal. In this thesis, MLP type is used to predict software effort based on Software size calculated based on the FPA method, Team Size and cumulative weight of metrics which affect the effort in agile environment [17].

Chapter 4: Proposed Methodology

In this chapter we will describe the underline framework for our methodology. There are two basic objectives for performance of this methodology. First is to identify all project characteristics along with their intensity and weights as they play a vital role in project estimation. Second is to use machine learning algorithm so that there is no ambiguity in project estimation. We propose to use ANN as they have been successfully applied in estimation of traditional methodology.

4.1 Project characteristics metrics

This Section states the description of different project characteristics metrics like complexity, team, programmer's capability etc. which are going to affect the overall effort required to develop software's through agile methodologies.

After studying the structure of different agile methodologies and different project estimation techniques proposed by different researchers shown in chapter 1, we have identified following effort metrics which are directly involved in effort estimation for agile methodology. The metrics contains different components which can affect the effort, some of these components are derived from traditional estimation methods like COCOMO2. These metrics can be broadly classified as:

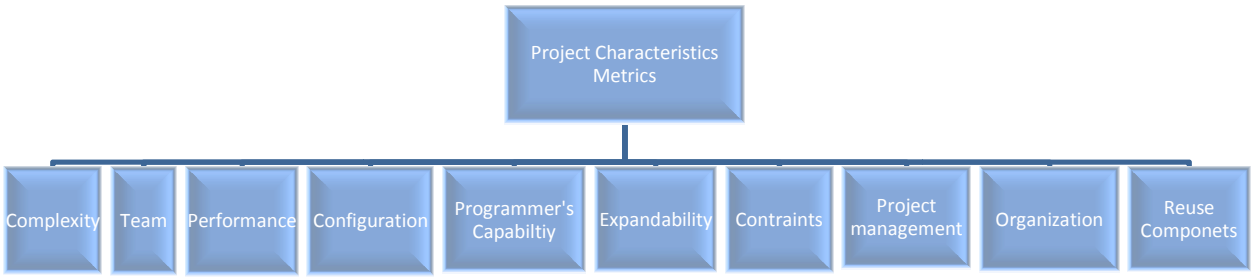

Figure 6: Project Characteristics Metrics

Complexity

Complexity (or **software complexity**) is a term that encompasses numerous properties of a piece of software, all of which affect internal interactions. Higher levels of complexity in software increase the risk of unintentionally interfering with interactions and so increase the chance of introducing defects when making changes.

- **No. of Iterations**: Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames that typically last from one to four weeks. Multiple iterations might be required to release a product or new features. Therefore more number of iterations will lead to higher complexity and vice versa.

- **No. of Story Card**: A story card represents a specific business requirement, typically a small-business functional component, which is estimated by the software development team to take less than 2 weeks (usually much less) of work effort for a pair of software developers.

- **Customer Involvement**: Agile methods place a strong emphasis on customer feedback and interaction. Projects with involved customers have much higher chances of success than projects which lack customer interaction.

- **Clarity of Requirements**: A common cause of IT project time and cost overrun is the impact of requirements issues on systems, where the expected system behavior is not clear.

Team

A project team is a team whose members usually belong to different groups, functions and are assigned to activities for the same project. A team can be divided into sub-teams according to need.

- **Pair Programming:** Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently.

- **Skill variety in Team:** It shows the versatile nature of the team which will be assigned to project. A team which posses a good tradeoff between testers, programmers is set to have high Skill variety and vice versa for low team skill variety.

- **Team Experience:** It is the overall experience of a team which will be working on the desired project. If a team is newly formed than it posses' lack of team experience on the other hand if a team has done enough number of projects together then the experience will be close to high.

- **Training required:** Training is the foundation of any software development as their might be some new technologies or tools will be used for the development so if in the team there is lack of experience in particular language or tool then the training required would be high and vice versa.

Performance

Project performance may be characterized as execution time, designing and coding standards, accuracy in outcome etc. as per customer requirements. Thus, the necessary features must be included in design and architecture of the software for achieving the performance level. These features of project increase the cost and duration of the project.

- **Load:** The system load is a measure of the amount of computational work that a computer system performs. It signifies the amount load which the software can expect while executing. If the estimation of load is high then its value would be high too and vice versa.
- **Execution time:** The execution time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services on its behalf.
- **Response time:** Response time is the time a system or functional unit takes to react to a given input. If the time needed is less than it will be a low for this factor and vice versa.
- **Number of queries:** It shows the amount of queries which are expected on the system when it is executing. If the number is high then this factor would have high value and vice versa.

Organization

A social unit of people that is structured and managed to meet a need or to pursue collective goals. All organizations have management structures that determines relationships between the different activities and the members, and subdivides and assigns roles, responsibilities, and authority to carry out different tasks.

- **CMM Level:** The Capability Maturity Model (CMM) is a development model created after study of data collected from organizations that contracted with the U.S. Department of Defense**.** There are five levels of it from 1-5, 1 signifies very low while 5 signifies very high.

- **Level of distraction in office:** This factor represent frequency of distraction during the work. If the distraction is minimum then its value would be very low and vice versa.

- **Multi Site Development:** This factor shows that the undergone project will fully developed at a particular site or on multiple site. If the system will be developed at multiple site then the time consume by the development process will be high as it will take time to integrate the different modules also here dependency will increase the time.
- **Management Involvement:** Management plays a great role by ensuring that the project gets over well within the time as well as in the budget. So if there is a strong coordination of management than the value of this factor would be Very High and vice versa.

Configuration

It is one of the key factors in Effort estimation. Configuration is in context of estimation refers to special hardware and software requirements to run the software smoothly.

- **Hardware Usage:** It depicts the overall use of hardware in the software on the scale of very low, low, medium, high and very high.

- **Complexity of Hardware:** This Represent the complexity of the hardware will be used in the software. More complex Hardware would take value very high and vice versa.

- **Response time of hardware:** It shows the response time of the hardware if the response is very fast then the value would be very high.

- **Hardware Changes:** It takes account of the hardware change that can occur later in the SDLC process. If there is need for it then the value for this factor would be on higher side and vice versa.

Programmer's Capability

This represents the capability of the programmers who will be working on the software product. The evaluation should not only consider the level of experience of the programmers (it is covered by other factors) and it should be based on the capability of the programmers as a team rather than as individuals.

- **Experience:** It represents the overall experience of a programmer. It can be categories as number of years example if the average experience of a programmer is between 6-10 years then it will get a value medium.

- **Platform experience:** It shows the overall experience of the programmer on a particular platform on which the software has to be built.

- **Tools experience:** It shows the overall experience of the programmer on a particular tool which will be used to develop the desired software.
- **Language experience:** It shows the overall experience of the programmer on a particular language by which the software has to be built.

Expandability

Ability of a computer system to accommodate additions to its capacity or capabilities or to increase the functionality.

- **Volatility of Requirement:** It represents the changing requirement of the software which indeed force the software to expend by increasing its functionality. So if the requirements are hugely volatile we assign this factor a value from high side.

- **Customer Feedback:** Customer feedback is most important factor which can expand the current domain to a much larger domain also it can help developers to find out the right functionality which the customer has not given earlier. A regular feedback will set its value to high.

- **Time to Integrate changes:** It is a measure of time which gives us an approximate about how much time one can need to integrate the changes.

- **Flexibility:** Modifying source code is quite easy, but it is very difficult to manage the impact of the changes on other part of source codes. Flexibility is the ease with which an operational program can be modified.

Reuse Components

We can reuse many things, for example, algorithms, designs, requirements specifications, procedures, modules, applications, ideas, design patterns, architectures. In this context it is required that reuse components can be easily combined with each other, especially without knowing from each other's existence.

- **Number of Reuse Modules**: In today's software development, related products and systems can be assembled from pre-built components. These reusable components can take a variety of forms, from existing software libraries, to free-standing commercial, off-the-shelf products (COTS) or open-source software (OSS), to entire software architectures and their components. Thus if the number of reuse module is high then the time taken to develop a software reduces by huge factor.

- **Cohesion**: it is a measure of how strongly-related each piece of functionality expressed by the source code of a software module is.

- **Efficiency of Reuse components**: At the same time efficiency of the reuse components are also important as if the reuse component is creating problem in integrating with

the current one then it would simple increase the overhead so a high value is desirable for smooth functioning.

- **Coupling**: coupling or dependency is the degree to which each program module relies on each one of the other modules. Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability.

Constraints

Software lives within the context of the real-world, and the real-world has constraints. This section allows you to state these constraints so it's clear that you are working within them and obvious how they affect your architecture decisions. Constraints are typically forced upon you, usually by the organization or customer that has asked for the software system to be built. Without constraints, there are often an infinite number of ways to solve the problem.

- **Time to market:** It gives us the idea about the time to deliver the first running iteration. If the time to market is very less then this factor will take a value on a higher scale and vice versa.

- **Size of data:** It represents the size of data. If the size of data is quite big then the factor will possess a high value and vice versa for small size of data.

- **Reliability:** Reliability is the ability of a system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances. If the system will be highly reliable then it will have a value on higher side and vice versa for lower reliable systems.

- **Risk Involved:** Risk is future uncertain events with a probability of occurrence and a potential for loss. Risk identification and management are the main concerns in every software project. Effective analysis of software risks will help to effective planning and

assignments of work. The more is risk involved in the project the more would be its value.

Project management

Project management is the discipline of planning, organizing, motivating, and controlling resources to achieve specific goals. In practice, the management of these software systems is often quite different, and requires a good development of distinct technical skills and management strategies.

- **Project Planning:** Project planning is part of project management, which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment. If the planning is good then the value for this factor would be high.

- **Project Tracking:** Project Tracking refers to the management of projects, which includes but is not limited to measuring and reporting the status of milestones, tasks and activities required in achieving the pre-defined project results.

- **Project Monitoring:** To work to its full potential, any kind of project needs to set out proposals and objectives. Then a monitoring system should be worked out to keep a check on all the various activities, including finances. This will help project staff to know how things are going, as well as giving early warning of possible problems and difficulties.

- **Documentation:** Software documentation or source code documentation is written text that accompanies computer software. It either explains how it operates or how to use it, or may mean different things to people in different roles. If there is need of large number of documents to support your work then this parameter will have higher values.

4.2 Proposed methodology using ANN

This section will demonstrate a framework which the new effort estimation methodology will be using in order to predict the most accurate effort value for particular software. Below Flow chart shows the overall framework for proposed methodology:

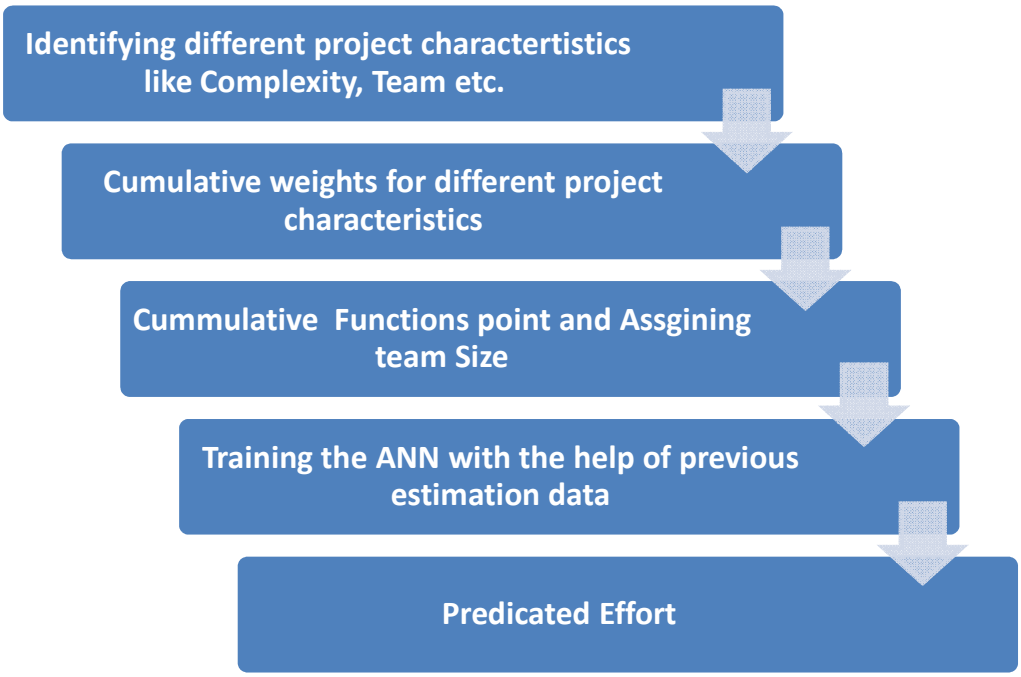

Figure 7: Proposed methodology

Software effort estimation process in any software project is not only essential, but also a very critical component. The success or failure of projects depends heavily on the accuracy of effort and schedule estimations, among other things. The emergence of agile methods in the software development field has presented many opportunities and challenges for researchers and practitioners alike. One of the major challenges is effort estimation for agile software development. Though traditional effort estimation approaches are used to estimate effort for agile software projects but they mostly result in inaccurate estimates. This research focuses on development of effort estimation model for agile software projects. Construction and use of the model is explained in detail. The model was calibrated using the empirical data collected

from 100 software projects. The experimental results show that model has good estimation accuracy in terms of MRE and Regression.

The proposed methodology is as follows:

1. **Identifying different project characteristics:**

   In section 4.1 we have defined various project characteristics which will affect the effort required to develop software's. These are some of the important project characteristics like complexity, team, performance etc which can be helpful for increasing accuracy of the effort estimation technique which the current technique lacks. All these characteristics are project driven and the project manager has to select the key project characteristics based on the functionality required. It might be possible that some of the project characteristics might not be available due to the functionality of the software, so the value of those characteristics will be Nominal (medium) by default. All this metrics are formed by taking care of each and every perspective of the software which can affect the cost and time.

2. **Cumulative weight of Project characteristics metrics**

   In this project manager will decide the weight age of different metrics by assigning different intensity level to the factors in the metrics. These weights would be calculated on the basis of certain criterion according to the project which is shown in appendix 1. The weight here will signify the weight age or the dominance of one metric over other. Finally adding the entire weights one can get cumulative weight, which is one of the input to the ANN which is going to be used to predict the effort. The value of cumulative weight will varies from 8 to 40. Following are the steps to find out the cumulative weights for different invented metrics.

   - **Identification of Intensity level**: There are total 10 Project Characteristics metrics (shown above) and each metrics contain four vectors which can affect the effort of

software development. The values of these vectors will be decided by Project Manager according to the requirement of the project. The values are decided on some criteria which is been shown in Appendix 1 for individual vector. To identify the intensity level of different vectors in different metrics a tool in PHP has been developed which gave user a GUI to select different intensity level on the grade of very low, low, medium, high and very high.

- **Weight calculation for each metrics**: Once the intensity level of different vectors are been identified, then through some general algorithm (summation) the weight of individual metrics are found. This value will depict the importance or the dominance of the metric in the effort required for building software.

- **Adding all the weights to get cumulative weight**: Now to get a cumulative or overall weight of the metrics, all the individual weights of 10 different metrics are added which gives us a value between 8 and 40. This cumulative weight will be the vital input to the ANN. This weight tells about the nature of the project, the more its value the more critical is project and vice versa.

$$\textbf{Cumulative weight} = \sum_{i=1}^{10} \textbf{weight}_i$$

Where $weight_i$ is the weight of i^{th} metric

Example: Complexity metrics contains different project characteristics like No. of iteration, No. of story points, Customer Involvement and Clarity of requirements. The values of these characteristics are assumed as:

No. of Iterations: Medium (0.6)

No. of story points: Low (0.4)

Customer Involvement: High (0.8)

Clarity of requirement: Medium (0.6)

Weight of this metrics will be 2.4.

Similarly the weights of the other metrics can be found. Adding all such weights will give us the cumulative weight.

3. **Cumulative Function point and assigning team size:**

As discussed earlier in chapter 3, the function point is the new way to estimate the software size. Unlike before it's hard to predict the estimated KLOC of the software now. So function point provides an estimate to software size. As we all know that the agile methodology decomposes the whole project into number of stories and in each iteration a set of users stories are been deployed. Generally a user story consists of an abstract description of the functionality to be developed. More the number of stories greater the size of software. So here function points are being calculated for each story and at the end each of the functions points are added to get the final cumulative function point for the overall project whose effort need to be calculated. Following are the guidelines for getting function point:

- **Identification of inputs and outputs on the basis of complexity**: The five function types identified are as external input, external output, external enquiries, internal logical files and external interface files and based on their complexity the values of these five function type is identified.

- **Calculating the unadjusted function point of each story**: Based on the user functionality and system characteristics identified above, we can find out the unadjusted function point for each story with the help of :

  Unadjusted Function Points = (User Functionality) X (System Characteristics)

- **Calculating the Adjustment factor of each story**: Here an adjustment factor is calculated for each story, which is product of the 14 system characteristics listed in chapter 3 on the scale of 1-5.

- **Getting the Adjusted Function point (AFP) for each story**: Now once we have both the adjustment factor and the unadjusted function point for each story, then we can calculate the final adjusted function point by:

  Adjusted Function Point (AFP) = (Unadjusted Function Point) X (Value Adjustment Factor)

- **Adding all the AFP to get cumulative function point**: Finally the Adjusted Function Point of each and every story will give us cumulative function point. This will be another input to the ANN. Formula for getting cumulative function point.

$$\text{Cumulative function point} = \sum_{i=1}^{k} AFP_i$$

Where AFP_i is the Adjusted function point of i^{th} story and number of stories are K

Based on the kind of development and types of requirement the Project manager will decide the maximum size of team which will be assigned for the project. This team size would be another factor which can affect the effort required to develop software's. So this will also be the critical input to ANN.

4. Training the ANN

Artificial Neural Network as discussed in chapter 3 is used for modeling complex relationships between inputs and outputs or to find patterns in data. Here to model the relationships between inputs and outputs a raw data which consist of the weights, function point, team size and effort of 100 old projects is passed to ANN, which will train the neural network to establish a relationship between the given set of inputs and output. The data set is collected from different sources and cumulative weights are being calculated based on the project characteristics. The data set is shown in Appendix 2. To create and train the ANN different tools (nftool, nprtool) from Matlab are used which are discussed in chapter 3.

ANN Model used for this research:

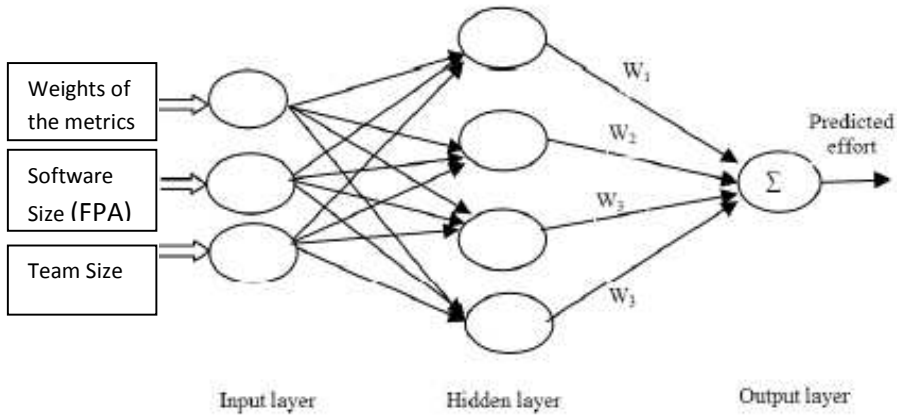

Figure 8: ANN Model

5. Effort Predication:

In this final step we will find out the effort required by passing different inputs to the well trained ANN. The inputs will be the cumulative weight, cumulative function point and the size of team which is going to be work on the project. All these inputs are identified in the above steps. The effort will be predicated on the basis of the values of inputs are been passed to ANN. While training the ANN, it establishes a relation between the inputs (cumulative weight, cumulative function point and team size) and the output (Effort). This relation will help one to predict the Effort by just providing the values of inputs. It has been found that the predicated effort poses less inaccuracy which indeed will help us to develop software's with high quality and well with in time and cost.

Chapter 5: Implementation

5.1 Tools used

This section states about the tool which will be going to be used for implementing Artificial Neural Network:

MATLAB (**mat**rix **lab**oratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran [24].

Neural Network Toolbox

It contains a set of MatLab functions which implement architectures and learning algorithms for several types of neural networks. The user can specify the architecture, the activation functions, the connectivity, the learning algorithm. To get a first idea on the facilities offered by NN toolbox just use help nnet (it lists all functions related with neural networks).

There are several demos which can be activated by using nnd. The toolbox contains both general functions (as sim to simulate a network or adapt and train to train a network) and particular functions for given architectures (as newlin, newff, newrbf etc.). Any neural network implemented in NN Toolbox is a structured object having as components arrays, functions for simulation and training and control parameters.

There are some applications with GUIs:

- Pattern recognition: nprtool
- Pattern fitting: nftool
- Clustering: nctool

NFTOOL:

nftool (neural network fitting tool) provides a graphical user interface for designing and training a feedforward neural network for solving approximation (fitting) problems. The networks created by nftool are characterized by:

- One hidden layer (the number of hidden units can be changed by the user; the default value is 20)
- The hidden units have a sigmoidal activation function (tansig or logsig) while the output units have a linear activation function
- The training algorithm is Backpropagation based on a Levenberg-Marquardt minimization method.

The learning process is controlled by a cross-validation technique based on a random division of the initial set of data in 3 subsets: for training (weights adjustment), for learning process control (validation) and for evaluation of the quality of approximation (testing). The quality of the approximation can be evaluated by:

- Mean Squared Error (MSE): it expresses the difference between to correct outputs and those provided by the network the approximation is better if MSE is smaller (closer to 0)

- Pearson's Correlation Coefficient (R): it measures the correlation between the correct outputs and those provided by the network; as R is closer to 1 as the approximation is better.

5.2 Interfaces of different module

This Section will demonstrate how the proposed methodology will be actually used with the help of tools.

1. Weights calculation

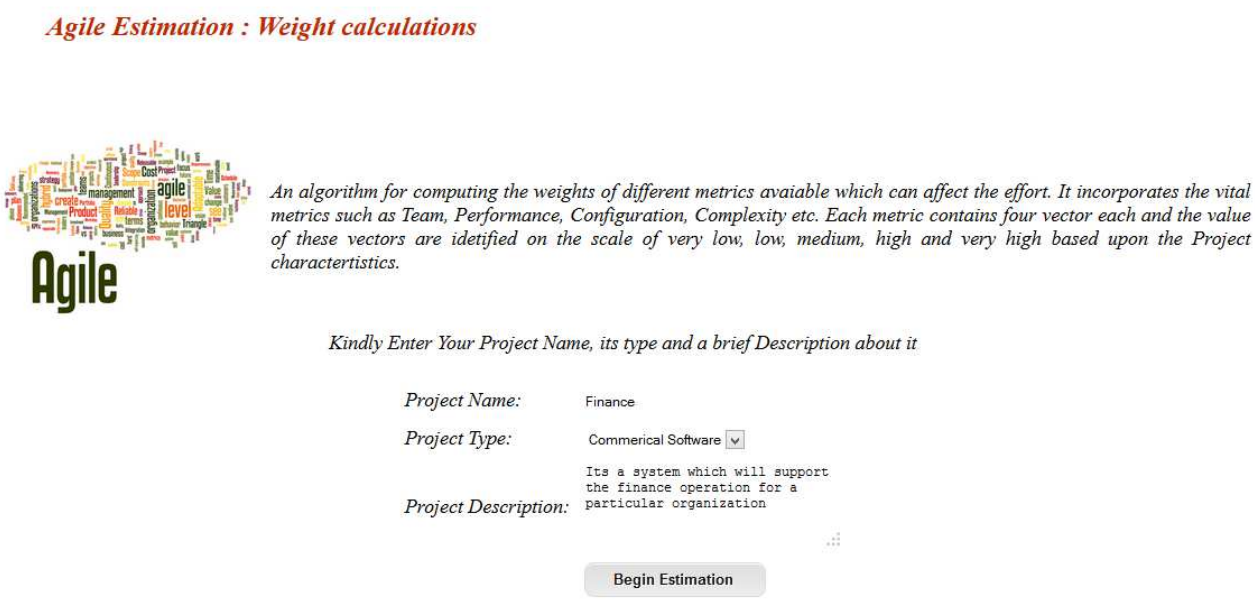

Figure 9: Starting of metrics Weight Calculations

Selecting the values of different metrics

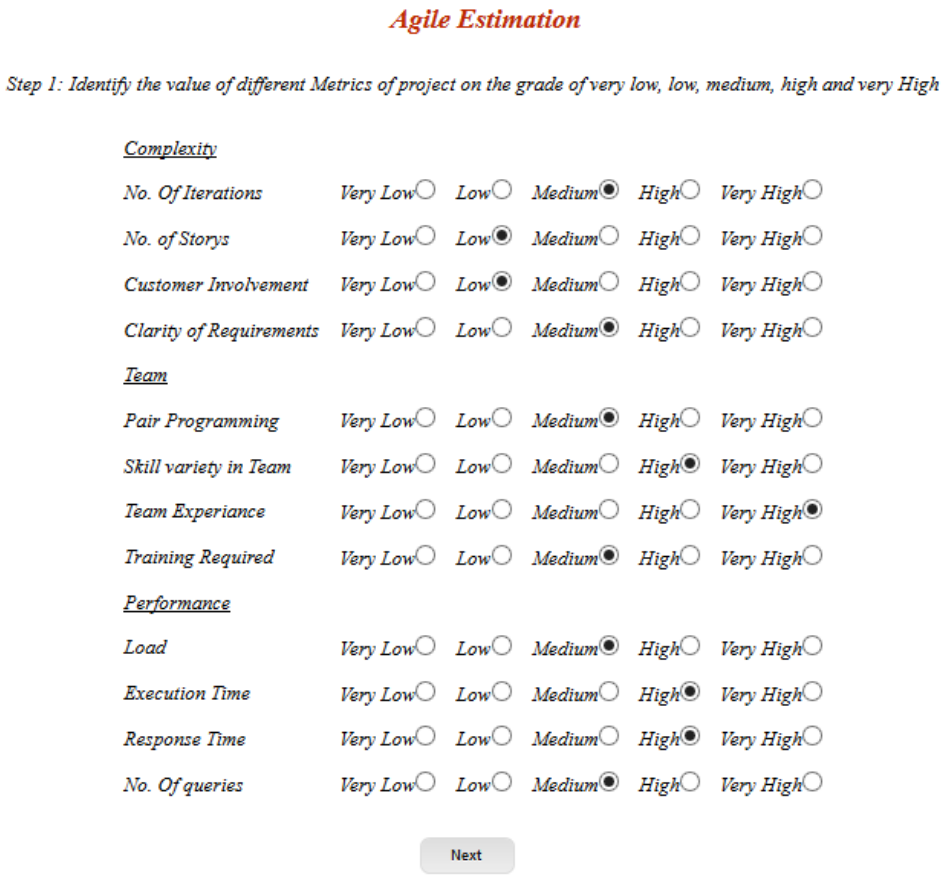

Figure 10: Selecting different intensity level of different metrics.

Selecting the values of different metrics

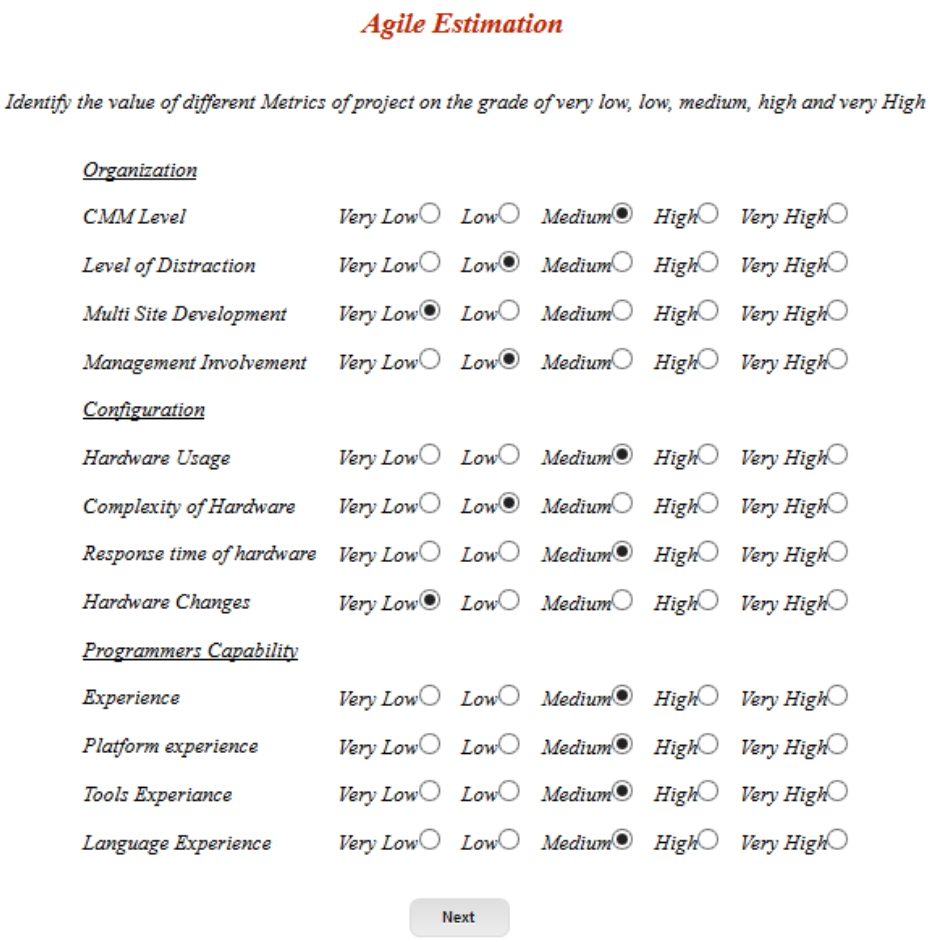

Figure 11: Selecting different intensity level of different metrics

Selecting the values of different metrics

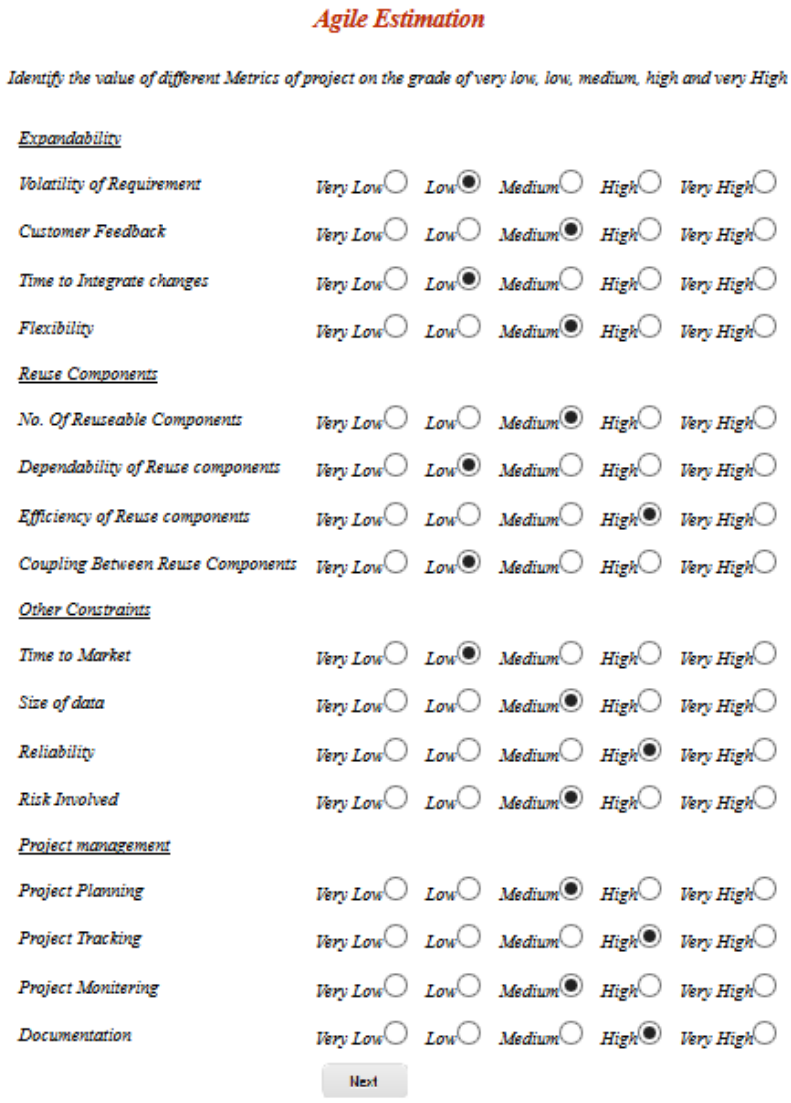

Figure 12: Selecting different intensity level of different metrics

Final weight calculations

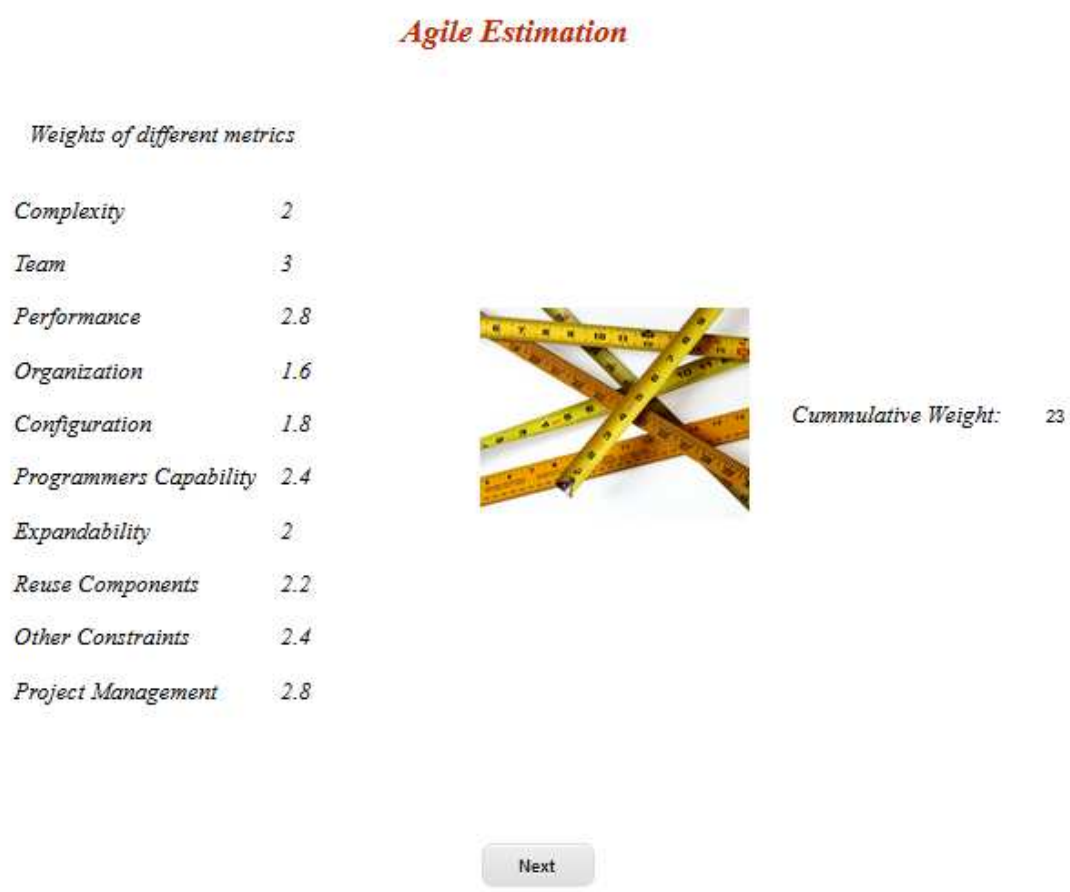

Figure 13: Final weights of different metrics and Cumulative weight

Training the ANN

Step 1: use command nftool in the command line of Matlab

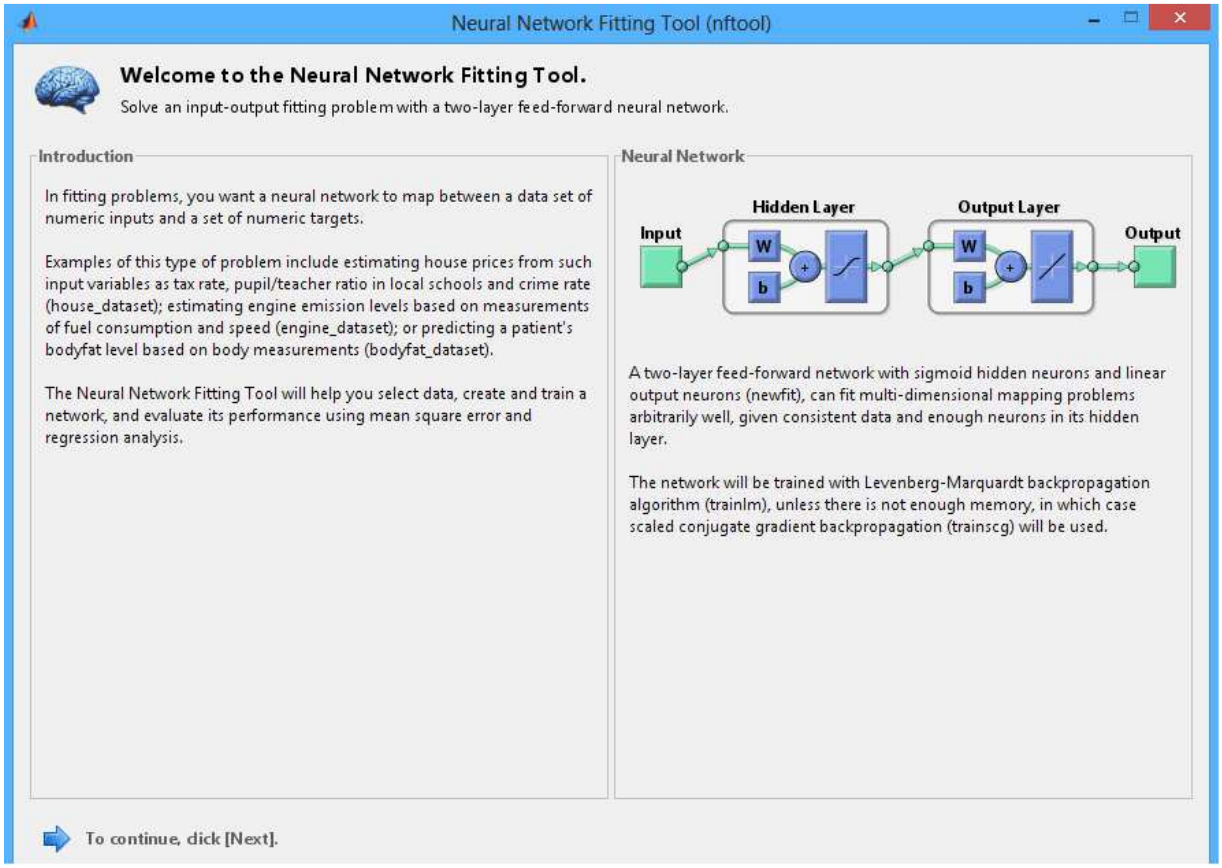

Figure 14: Starting nftool

Step 2: Selecting input and output parameters

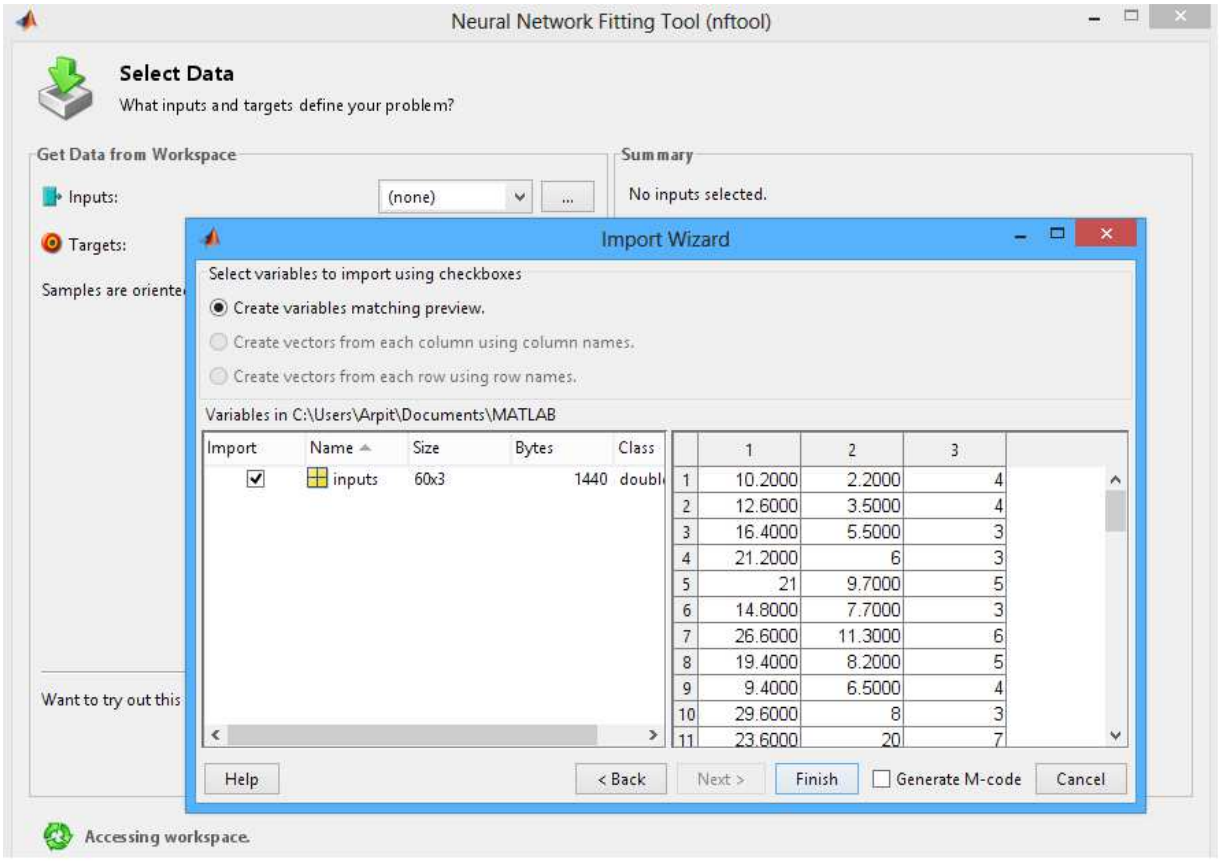

Figure 15: Input and output of ANN

Step 3: Assigning percentage of data for validation and testing

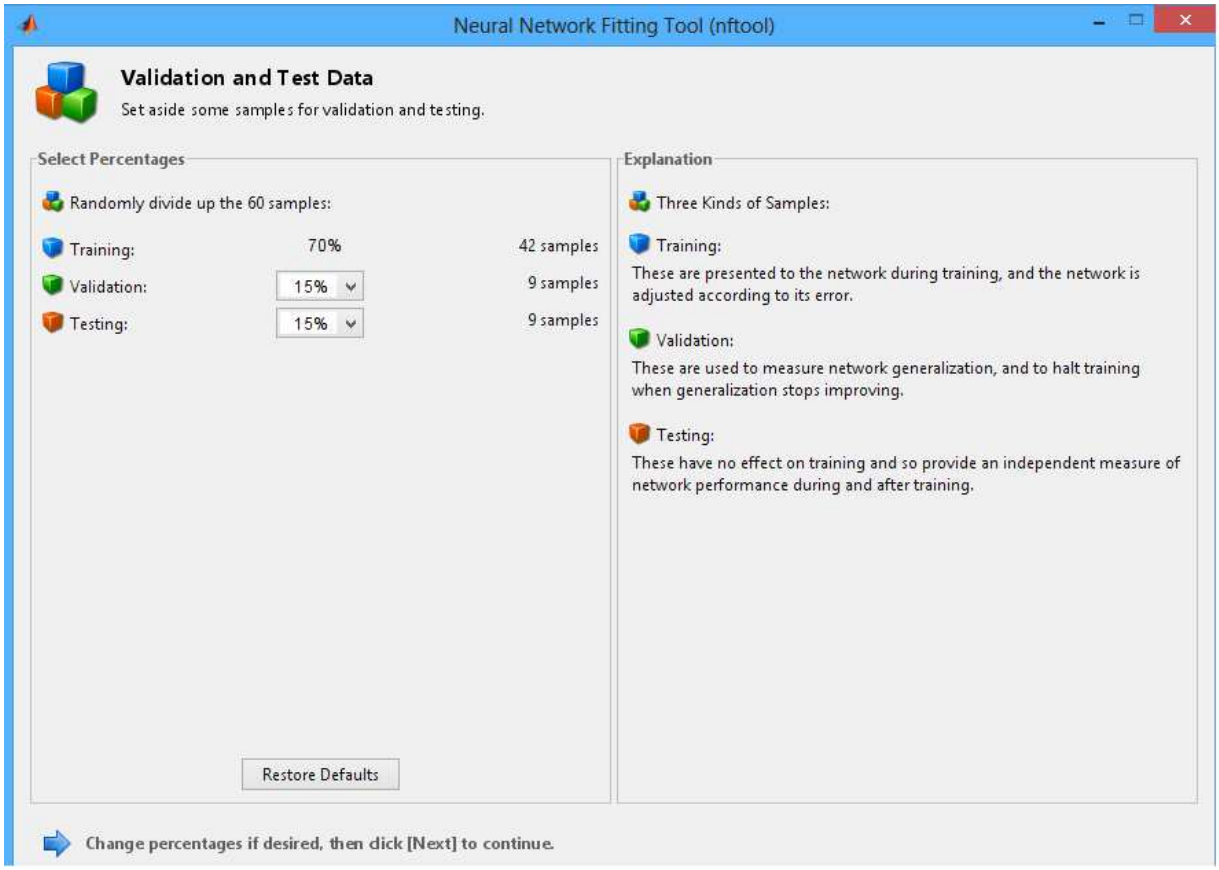

Figure 16: Sample Classification

Step 4: Selecting number of neurons for the hidden layers

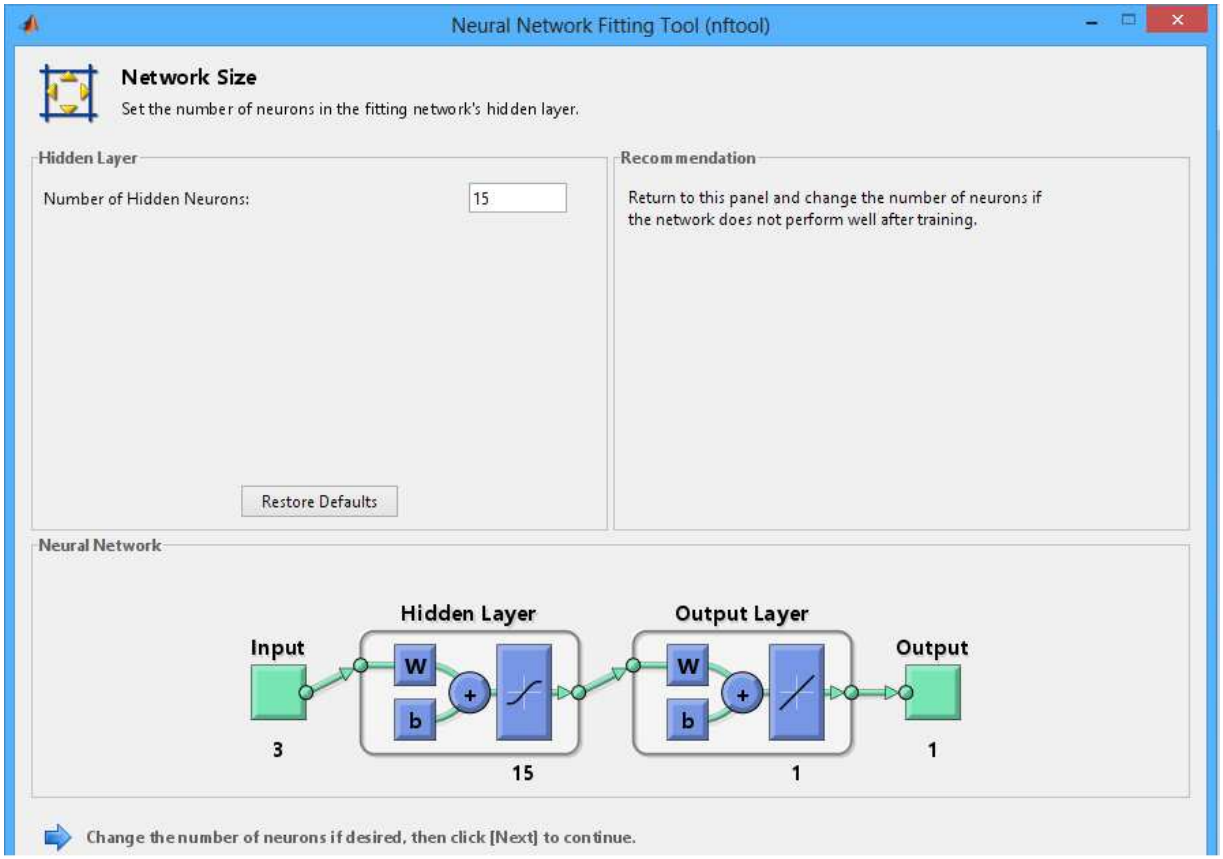

Figure 17: Selecting Number of Neurons

Step 5: ANN is ready for training. Just need to press train button

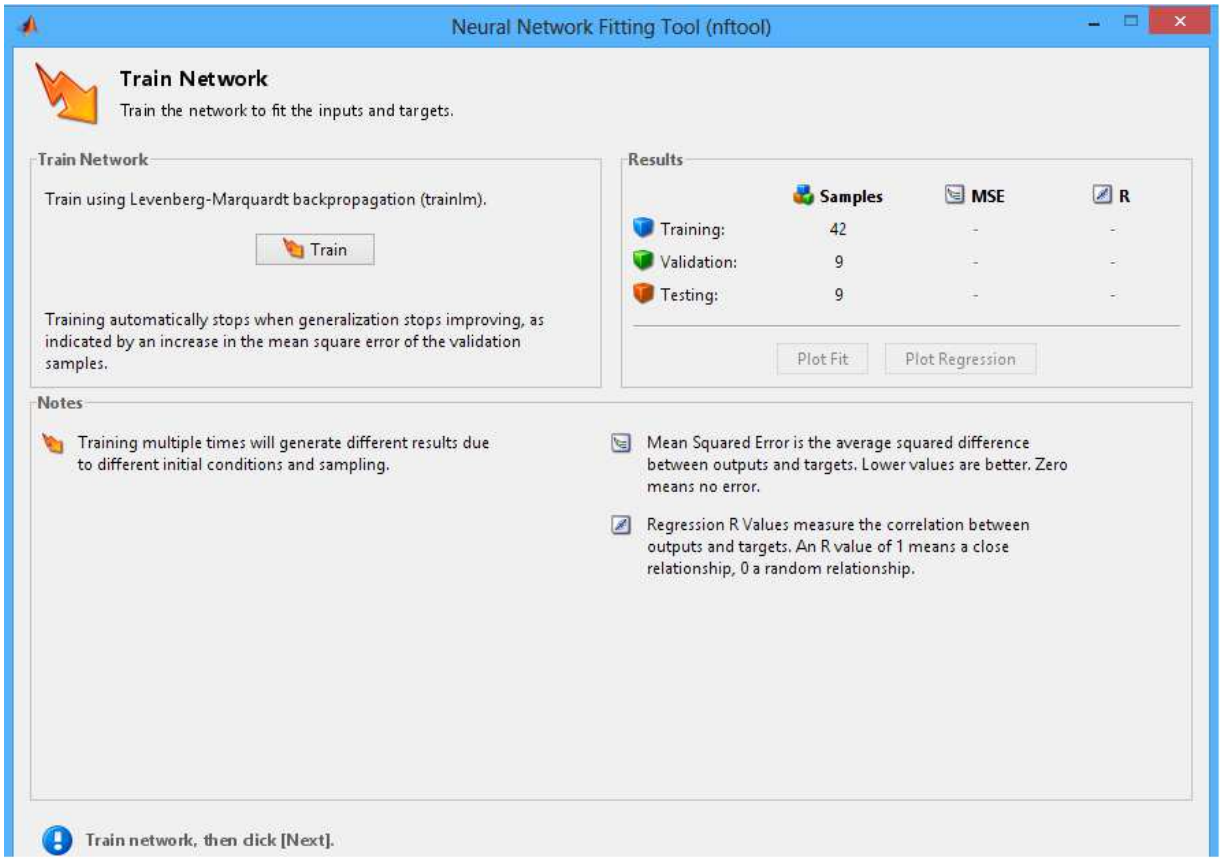

Figure 18: Training the Network

Step 6: Output of training the ANN

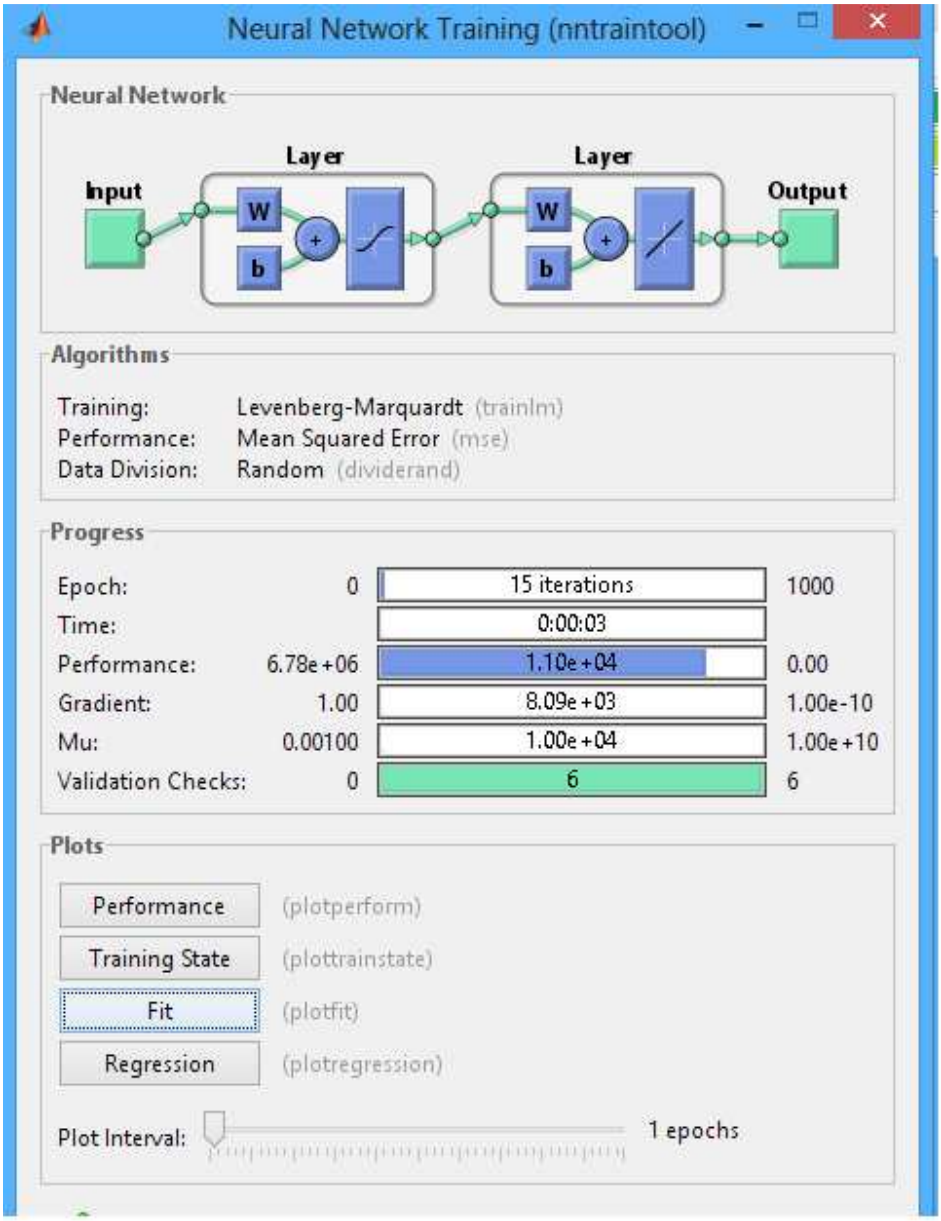

Figure 19: Results of ANN

Step 7: Saving the results and generating M file for the ANN.

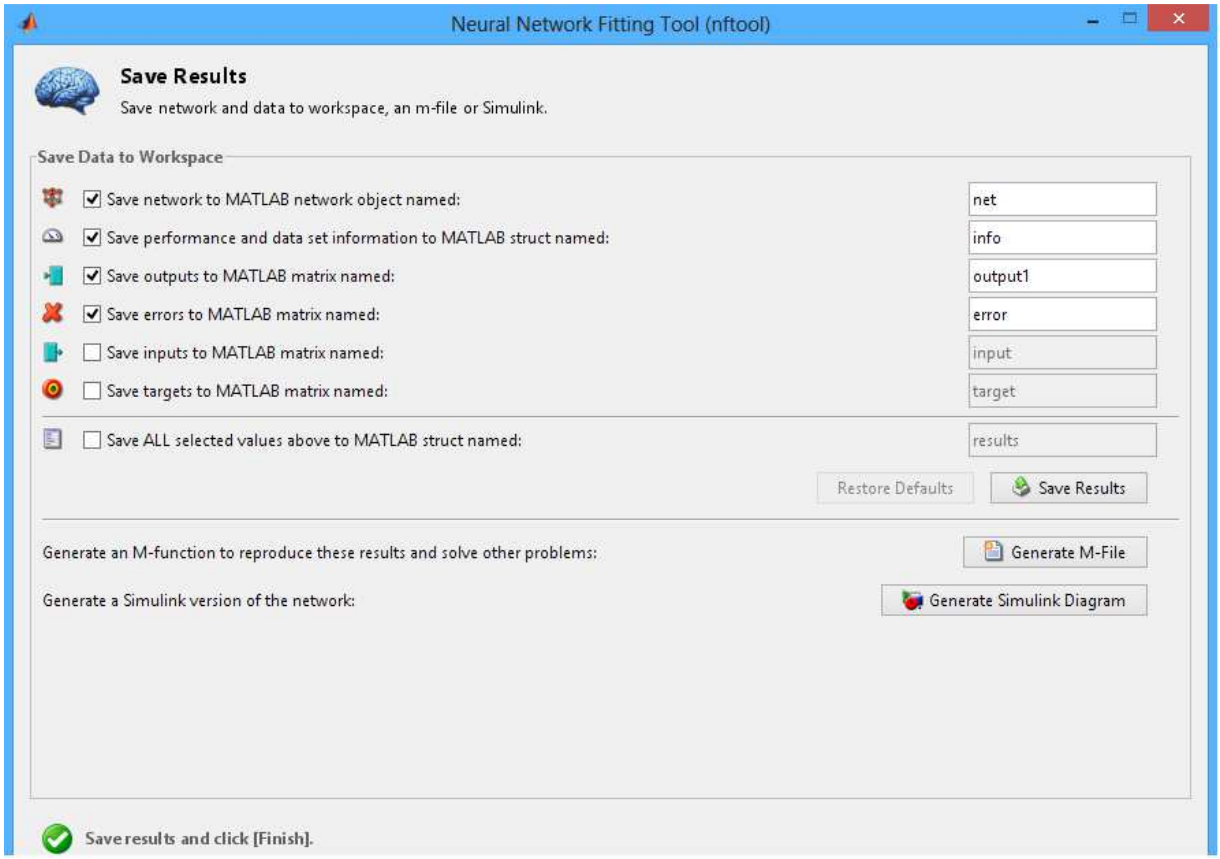

Figure 20: Saving results

Chapter 6: Results and Analysis

This section shows the results of above proposed methodology and the analysis of the result to check whether it can serve our purpose or not.

6.1 Over Estimation and Under Estimation

Problems with over-estimation

Managers and other project stakeholders sometimes fear that, if a project is overestimated, Parkinson's Law will kick in—the idea that work will expand to fill available time. For example, if you give a developer 5 days to deliver a task that could be completed in 4 days; the developer will find something to do with the extra day. As a result, some managers consciously squeeze the estimates to try to avoid Parkinson's Law [11].

Problems with under-estimation

Reduced effectiveness of project plans

Low estimates undermine effective planning by feeding bad assumptions into plans for specific activities. If the estimation errors caused the plans to be off by only 5% or 10%, those errors wouldn't cause any significant problems. But numerous studies [11] have found that software estimates are often inaccurate by 100% or more. When the planning assumptions are wrong by this magnitude, the average project's plans are based on assumptions that are so far off that the plans are virtually useless.

Poor technical foundation leads to worse-than-nominal results

A low estimate can cause you to spend too little time on understanding requirements. If you don't put enough focus on understanding them, you'll have to revisit them later in the project at greater cost than if you had done that well in the first place. This ultimately makes your project take longer than it would have taken with an accurate estimate [17].

Destructive late-project dynamics make the project worse than nominal

Once a project gets into "late" status, project teams engage in numerous activities that they don't need to engage in during an "on-time" project. For example, more status meetings, frequent re-estimation, defects arising from quick and dirty workarounds etc.

Over-estimation vs. Under-estimation

As Figure shows, the best project results come from the most accurate estimates. If the estimate is too low, planning inefficiencies will drive up the actual cost and schedule of the project. If the estimate is too high, Parkinson's Law kicks in.

Figure 21: Penalties for underestimation vs. Penalties for overestimation [14]

Work does expand to fill available time. But deliberately underestimating a project because of Parkinson's Law makes sense only if the penalty for overestimation is worse than the penalty for underestimation. In software, the penalty for overestimation is linear and bounded - work will expand to fill available time, but it will not expand any further. But the penalty for underestimation is nonlinear and unbounded - planning errors, shortchanging understanding of requirements, and the creation of more defects cause more damage than overestimation does, and with little ability to predict the extent of the damage ahead of time [2][11][12].

6.2 Analysis using a Data set

The data set which has been used to train the ANN is shown in Appendix 2. The data set is the famous china set which is widely used for validating the methodology to find out the effort required to develop software. The data set consist of various attributes like function point, development types, actual time taken, actual effort etc for around 100 projects. Also from the attribute of data set one can easily find the project characteristics and thereby can have a good idea about the project. This may help one to figure out weights of the metrics which indeed will give us back the cumulative weight.

6.2.1 Selection of inputs

The inputs to the ANN are selected on the basis of their quality to accurately identify the effort required. The input must be in direct correspondence with the effort.

Following are the selected inputs:

1. Cumulative weight of the different metrics which can affect the effort.
2. Cumulative function point of the different stories.
3. Maximum Team size

6.2.2 Evaluation Criteria

The evaluation criteria for the proposed methodology is regression values and MRE (Mean relative error), the ideal values of the regression is close to one (more closer the value to one, more accurate results) and zero of the MRE (more closer the value to zero, more accurate result).

Different error measurements have been used by various researchers. I have choose the mean relative Error (MRE) and regression as the major measurement tool:

MRE: MRE is calculated as follows:

$$MRE = \frac{|\text{Actual Effort} - \text{Estimated Effort}|}{|\text{Actual Effort}|}$$

Where estimate is the network output for each observation and actual is the target value given, and n is the number of observations. If MRE is small, then the better is the model and the predictions are a good set of predictions. The MRE value is calculated for each observation whose EFFORT is predicted. The MRE is sensitive to individual predictions with some observations having excessively large MREs. Therefore another measure namely median of MRE values can be used. (MdMRE)[18].

Another criterion that can be used is prediction at level l, PRED (l) =k/n, where k is the number f observations where MRE is less than or equal to l.
For example: PRED (.1) =.75 means that 75% of the cases have estimates within 10 % of their actual values.

Regression: Regression analysis is a statistical technique for estimating the relationships among variables. More specifically, regression analysis helps one understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed. Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. If regression value is close to one, then the better is the model and the predictions are a good set of predictions.

6.3 Results

The result of the well trained ANN will be the graph which shows different regression values during training, validation and Testing. It has been observed that changing the number of hidden layers in ANN or changing the number of neurons in the hidden layer of ANN may affect the regression values. So for good approximate results the training of ANN has to be done several times by changing the hidden layers. So the testing has been done by changing the hidden layer many times from 1 to 35 but the most significant values of regression is obtained on 15.

Below shown the regression plot of the data set shown in appendix-2 which is being used to train the ANN with different number of neurons in the hidden layers.

Number of neurons in hidden layer is 15 and the regression value while testing the trained ANN is around 0.97 that means around 0.03 is error value.

Training:

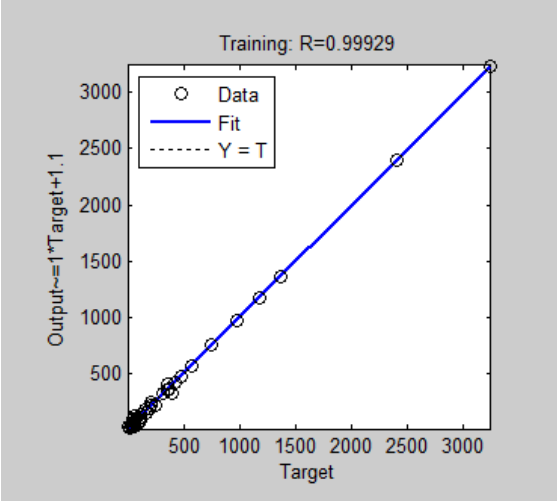

Figure 22: Regression Curve of Training

Validitaion:

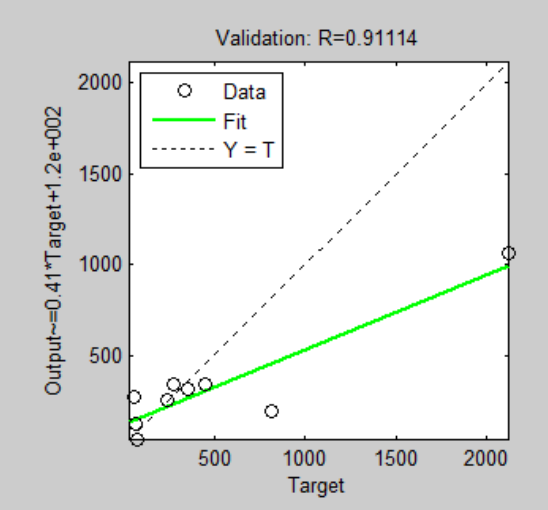

Figure 23: Regression Curve of Validation

Testing:

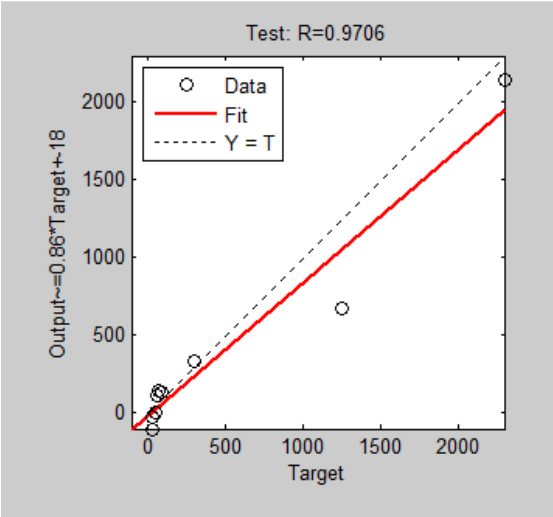

Figure 24: Regression Curve of Testing

Overall:

Training	Validation	Testing
0.99929	0.91119	0.9706

Table 3: Collective values for Regression Curve for 15 hidden neurons

Number of neurons in hidden layer is 10 the regression value here is around 0.96 which is less than the above regression value:

Training:

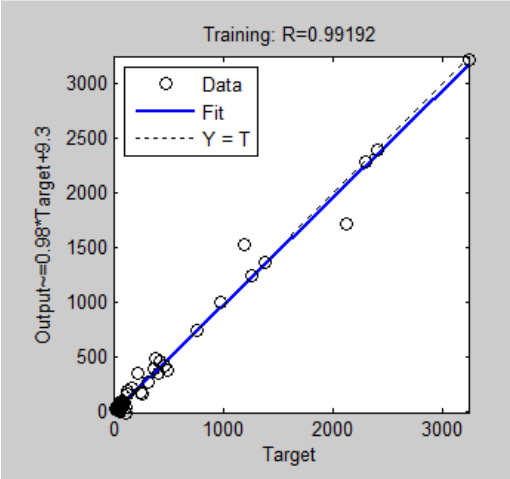

Figure 25: Regression Curve of Training

Validation:

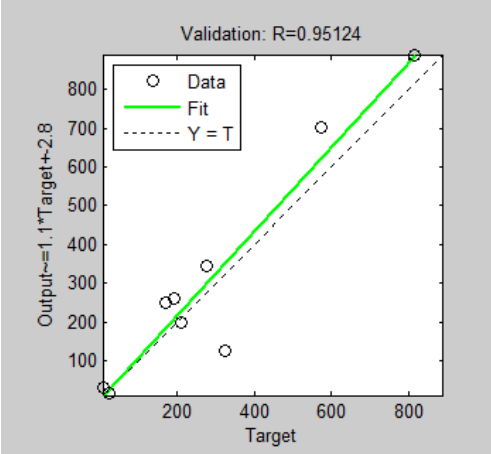

Figure 26: Regression Curve of Validation

Testing:

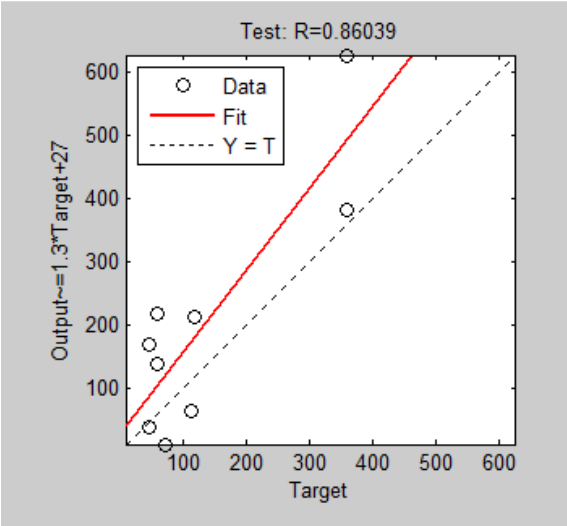

Figure 27: Regression Curve of Validation

Overall:

Training	Validation	Testing
0.99192	0.95124	0.86039

Table 4: Collective Values of Regression Curve for 10 Hidden neurons

So comparing table 3 and table 4, one can easily indentify that the better values of regression are in table 3 which is close to 0.97 that means the Relative error comes out to be 0.03 and this relative error is the minimum one till date with respect to all the effort estimation methodologies using ANN. So here I can say that it has been shown that on the basis of 100 data sets that the ANN can also be very efficient for effort estimation for a project which are developing through agile techniques as well.

Results can be stated as follows:

- **Decreased MRE and increased the estimation accuracy**

  Magnitude of relative error is a widely used measure for estimation accuracy.

  It is defined as:

$$MRE = \frac{|\text{Actual Effort} - \text{Estimated Effort}|}{|\text{Actual Effort}|}$$

  Proposed methodology was able to decrease the estimation error, which naturally increased the accuracy of estimation, in comparison with other available methods like planning poker. The result has shown that MRE values are reduced up to 0.03%.

- **Reduced losses due to estimation inaccuracy**

  With the application of proposed methodology and analysis, one can be able to show that in the long run, we can reduce the losses incurred due to estimation inaccuracy, i.e. under-estimation and over-estimation.

- **Naturally fitting method for agile development**

  We have also identified the next steps to validate and adopt proposed methodology in order to improve the estimation process. The idea is to train ANN with the data sets of agile projects of previously developed projects estimate data in the first part of the project, and get more precise results in the latter half.

Chapter 7: Conclusion

This thesis proposed a feed-forward Artificial Neural Network (ANN) model to predict software effort in agile environment based on the different project characteristics metrics. The inputs of the proposed model are software size which is function points, Cumulative weights of 10 metrics which contain vital factors which can affect the effort required to develop software's and maximum team size. To evaluate the ANN model, a multiple linear regression model was developed that has the same inputs as the ANN model. The regression based ANN models was trained using 100 projects and evaluated using 40 projects. The ANN model was then evaluated against the regression and produced great results which are shown above.

Results show that the proposed ANN model for agile estimation outperforms the existing methods for agile effort estimation based on the MRE and Regression values criteria and can be used an as alternative method to predict software effort in agile environments.

Advantages of New Technique:

- **An expert independent method:** The estimation model which uses ANN is independent of the experts. That means ANN has removed the role of the experts from Estimation process which can be very helpful to produce accurate results.
- **Lower MRE values in estimation:** This technique has shown a decent lower MRE value so one can expect to achieve a good accuracy while estimating effort.
- **Evolve with time:** As it uses ANN which can evolve itself with time, as the new projects are being developed, the data of the projects can be useful to train the neural network and thereafter can be useful to predict the effort and cost of new projects.

Future work will focus on trying other models such as Radial Basis Function Neural Network and General Regression Neural Network.

References

[1] Agile Modeling Home Page. Effective Practices for Modeling and Documentation. [Online] Retrieved 17th March 2009. Available at: www.agilemodeling.com .

[2] J. Erickson, K. Lyytinen and K. Siau, Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research. In Journal of Database Management, 16(4), 2005, 88-100.

[3] M. Singh, U-SCRUM: An Agile Methodology for Promoting Usability. In Ag. AGILE '08. Conference, Toronto, 2008, 555-560.

[4] M. Cristal, D. Wildt and R. Prikladnicki, Usage of SCRUM Practices within a Global Company. Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on, 2008, 222-226.

[5] Abrahamsson, P., Fronza, I., Moser, R., Vlasenko, J., Pedrycz, W., Predicting Development Effort from User Stories, International Symposium on Empirical Software Engineering and Measurement, 2011

[6] Haugen, N., an Empirical Study of Using Planning Poker for User Story Estimation, Proceedings of AGILE 2006 Conference, 2006

[7] Zivadinovic, J., Medic, Z., Maksimovic, D., Damnjanovic, A., Vujcic, S., Methods of Effort Estimation in Software Engineering, International Symposium Engineering Management and Competitiveness 2011 (EMC2011)

[8] Cohn, M., Agile Estimating and Planning, 2002

[9] Molokken, K., Jorgensen, M., A Review of Software Surveys on Software Effort Estimation, International Symposium on Empirical Software Engineering, 2003.

[10] Goldratt, E.M., Critical Chain, Great Barrington, MA: The North River Press, 1997

[11] Caruana, R., Niculescu-Mizil, A., An Empirical Comparison of Supervised Learning Algorithms, Proceedings of the 23rd International Conference on Machine Learning, 2006

[12] Cheng, B., Xuejun, Y., The Selection of Agile Development's Effort Estimation Factors based on Principal Component Analysis, International Conference on Information and Computer Applications, 2012

[13] Baskeles, B., Turhan, B., Bener, A., Software Effort Estimation Using Machine Learning Methods, 22nd International Symposium on Computer and Information Sciences, 2007

[14] C. Lopez-Martin, C. Isaza and A. Chavoya, "Software development effort prediction of industrial projects applying a general regression neural network," Empirical Software Engineering, vol. 17, pp. 1-19, 2011.

[15] C.W. Dawson."A Neural Network Approach to Software Projects Effort Estimation", TransactionS on Information and Communication Technology 1996.

[16] ISBSG – Repository Data Release 11

[17] A. B. Nassif, L. F. Capretz and D. Ho, "Software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model," in 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), Kyoto, Japan, 2012, pp. 589-594.

[18] Iman Attarzadeh, Siew Hock Ow. Proposing a New Software Cost Estimation Model Based on Artificial Neural Networks. International Conference on Computer Engineering and Technology, 2010, PP:487-491.

[19] L. Pickard, B. Kitchenham, and S. linkman, "An Investigation Analysis Techniques for Software Datasets," in Proc. of Sixth IEEE International Software Metrics, Symposium, 1999, pp. 1-13.

[20] Y. Zheng, B. Wang, Y. Zheng, and L. Shi, "Estimation of Software Project effort based on functional Point," in Proc. of 4th International Conference on Computer Science and Education , pp. 941-943, 2009.

[21] M. Sadiq and S. Ahmed, "Relationship between Lines of Code and Function Point and its Application in the Computation of Effort and Duration of a Software using Software Equation," in Proc International Conference on Emerging Technologies and Applications in Engineering, Technology and Sciences Rajkot, Gujarat, 2008, India.

[22] R Moser, W Pedrycz, G Succi Incremental effort prediction models in Agile Development using Radial Basis Functions Proc. 19th International Conf. on Software Engineering & Knowledge.

[23] Aggrawal KK , Yogesh Singh, Software Engineering 2nd Edition New age Publication.

[24] Matlab - www.mathworks.com

[25] Sommerville I. Software Engineering – 8th Edition

[26] Gupta D, Diwedi R, Kumar S, Domain Specific priority based implementation of mobile service – an Agile Way , International Conference of Software Engineering and Research, Las Vegas, USA.

[27] Coelho E, Basu A, Effort Estimation in Agile software Development using Story Points, International Journal of applied Information System, New York, USA Volume 3-No. 7 August 2012.

[28] Cagley Thomas, Agile Estimation using Functional Metrics- one of current thread of thought as voiced by Tim Lister on the Software Process and Measurement Cast 51.

[29] B. Tessem, Experiences in Learning XP Practices: A Qualitative Study. In Extreme Programming and Agile Processes in Software Engineering. 2003, 131-137.

Appendix 1

It shows the different criteria of selecting the intensity level of different vectors in the metrics:

Classification	Vital Factors	Description
Complexity	Number of Iterations	1-5: Very low 6-9 Low 10-14 Medium 14-20 High >20 very High
	Number of Story Cards	1-10: Very low 11-15 Low 16-24 Medium 25-30 High >30 Very High
	Customer Involvement	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Clarity of requirements	<20 %: Very low 20-30 % Low 31-50 % Medium 51-70 % High >70 % Very High

Classification	Vital Factors	Description
Team	Pair Programming	<20 %: Very low 20-30 % Low 31-50 % Medium 51-70 % High >70 % Very High
	Skill Variety in Team	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Team Experience	<1 year: Very low 1-3 year Low 3-5 year Medium 5-7 year High >7 year Very High
	Training Required	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Performance	Load	<20 % Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Execution time	>1 sec Very low 800-1000 msec Low 500-799 msec Medium 200-499 msec High <200 msec Very High
	Response time	>1 sec Very low 800-1000 msec Low 500-799 msec Medium 200-499 msec High <200 msec Very High
	No. of queries	<100 Very low 100-200 Low 200-400 Medium 400-500 High >500 Very High

Classification	Vital Factors	Description
Organization	CMM Level	Level 1: Very low Level 2 Low Level 3 Medium Level 4 High Level 5 Very High
	Level of distraction	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Multisite Development	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Management Involvement	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Configuration	Hardware usage	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Complexity of hardware	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Response time of hardware	>1 sec: Very low 800-1000 msec Low 500-799 msec Medium 200-499 msec High <200 msec Very High
	Hardware changes	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Programmer's Capability	Experience	<1 year: Very low 1-3 year Low 3-5 year Medium 5-7 year High >7 year Very High
	Tool Experience	<1 year: Very low 1-2 year Low 2-3 year Medium 3-4 year High >4 year Very High
	Language Experience	<1 year: Very low 1-2 year Low 2-3 year Medium 3-4 year High >4 year Very High
	Platform Experience	<1 year: Very low 1-2 year Low 2-3 year Medium 3-4 year High >4 year Very High

Classification	Vital Factors	Description
Expandability	Volatility of requirements	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Customer Feedback	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Time to integrate changes	<1 weeks: Very low 1-2 week Low 2-3 week Medium 3-4 week High >4 week Very High
	Flexibility	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Reuse Components	Number of Reuse components	<5: Very low 6-10 Low 11-13 Medium 14-17 High >18 Very High
	Cohesion	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Efficiency of Reuse components	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Coupling	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Constraints	Time to market	>1 year Very low 9-12 months Low 6-8 months Medium 2-5 months High <2 months Very High
	Size of data	<10 MB: Very low 10-30 MB Low 31-50 MB Medium 51- 70 MB High >70 MB Very High
	Reliability	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Risk Involved	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Classification	Vital Factors	Description
Project Management	Project Planning	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Project tracking	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Project monitoring	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High
	Documentation	<20 %: Very low 20-30 % Low 30-50 % Medium 50-70 % High >70 % Very High

Appendix -2

This shows the data set which is used to train the ANN. Following are the fields:

1. Cumulative Weights
2. Software Size
3. Team Size
4. Actual Effort

S.No	Cumulative Weights	Software Size(In FP)	Team Size	Actual Effort
1	10.2	1587	20	7490
2	12.6	260	15	4150
3	16.4	152	18	668
4	21.2	252	14	3238
5	21	292	25	2994
6	14.8	83	21	1333
7	26.6	79	14	1607
8	19.4	97	24	1158
9	9.4	116	21	1243
10	29.6	52	30	3372
11	23.6	465	14	10200
12	14.6	67	15	1704
13	24.4	199	21	2640
14	29	176	4	3348
15	22.8	391	5	676
16	17.6	263	5	911
17	20	42	8	2496
18	34.4	190	12	1171
19	26.2	245	12	3532
20	14.8	77	6	436
21	33.2	355	6	909
22	25.4	3156	9	9094
23	31	46	5	344
24	33	56	8	296
25	28	106	7	3503
26	29.2	71	6	246

27	22.8	306	9	2082
28	34.6	244	8	191
29	30.2	98	13	2974
30	24.8	331	7	328
31	26.2	101	9	406
32	27	192	20	1785
33	27.6	60	15	471
34	28	180	18	700
35	31.2	118	14	579
36	29.6	73	25	1211
37	31.8	143	21	1139
38	33	2190	14	14520
39	24.6	9	24	239
40	23.6	203	21	1262
41	18.4	162	30	1754
42	28.4	183	14	1514
43	25.6	59	15	667
44	22.2	412	21	5864
45	27.8	348	5	973
46	30.2	79	17	400
47	30.2	190	32	420
48	24.6	90	22	450
49	28	115.8	31	480
50	29.8	78	4	571.4
51	19.4	101	9	750
52	28.6	161.1	15	815
53	31	284.7	18	973
54	25.7	227	12	1181
55	27.2	177.9	14	1248
56	29.8	282.1	9	1368
57	27	219	10	2120
58	22	423	35	2300
59	34.2	302	23	2400
60	32.8	370	20	3240
61	29.2	562	27	5727
62	30.6	156	15	2040
63	24.6	75	18	3677
64	28	136	14	570
65	25.8	96	25	1223
66	21	78	21	976
67	29	51	14	387

68	31.6	134	24	286
69	27.8	1092	21	4416
70	24.8	75	30	305
71	18.4	88	14	129
72	32.4	3088	15	4266
73	30.2	82	21	440
74	30.2	110	23	1396
75	24.6	308	11	2533
76	28	143	15	225
77	29.8	82	28	440
78	19.4	73	20	2054
79	28.6	64	16	252
80	31	497	20	2362
81	28	288	15	2459
82	29.2	494	18	8706
83	22.8	130	14	770
84	34.6	204	25	8111
85	30.2	17	21	762
86	24.8	60	14	136
87	26.2	73	24	358
88	27	169	21	481
89	27.6	1351	30	3189
90	28	2087	14	22500
91	34.2	253	15	11719
92	32.8	102	21	183
93	29.2	115	32	1482
94	24.8	24.6	7	117.6
95	26.2	29.5	9	120
96	27	19.3	20	155
97	27.6	32.6	15	170
98	28	35.5	18	192
99	22.6	199	20	943
100	29.6	100	25	215