A Dissertation on

Cost Estimation Model for Agile Software Development Projects Using Exploratory Factor Analysis and Constraint Programming Approach

Submitted in the partial fulfilment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY

SAKSHI GARG

Roll No - 2K12/CSE/19

UNDER THE SUPERVISION OF

DR. DAYA GUPTA

Professor, Former HOD

Department of Computer Engineering, DTU

Department of Computer Engineering

Delhi Technological University, Delhi

2012-2014

DECLARATION

This thesis is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions.

The work was done under the guidance of Dr Daya Gupta (Professor, Former HOD, Department of Computer Engineering, DTU) at Delhi Technological University, Delhi

SAKSHI GARG
(Roll No- 2K12/CSE/19)
M.Tech CSE
Deptt of Computer Engineering
Delhi Technological University, Delhi

Acknowledgment

.

I would like to take this opportunity to express sincere gratitude to my mentor and project guide **Dr. Daya Gupta** (Prof., Former HOD, Deptt. of Computer Engineering, DTU). I am greatly indebted to her for her insightful and encouraging words that have been a driving force. I am extremely thankful to her for guidance.

I wish to convey my sincere gratitude to **Prof. Rajeev Kapoor**, Head of Department, and all the faculties and PhD. Scholars of Computer Engineering Department, Delhi Technological University who have enlightened me during my project.

I would also like to express my deep gratitude towards Dr. XuenJun Yu, Research scholar at the 'Centre for Systems and Software Engineering' at the 'School of Engineering, University of South California' for his valuable insight at the critical stages of the project.

Also I wish to express my gratitude towards Shruti Jaiswal, Research Scholar, Computer Engineering Department, DTU for her guidance and support during the making of this thesis.

I would also like to Lab staff for their timely cooperation help.

Lastly I would like to thank all who inspired and helped during the project.

SAKSHI GARG
(2K12/CSE/19)
M.Tech CSE
Deptt of Computer Engineering
Delhi Technological University, Delhi

Delhi Technological University

CERTIFICATE

This is to certify that the thesis titled "**Cost Estimation Model for Agile Software Development Projects**" submitted by **Sakshi Garg** (Roll No-2K12/CSE/19) of Final Semester, M.Tech. Computer Science and Engineering, Department of Computer Engineering, DTU, is a record of bonafide work undertaken by her under my supervision. The work fulfils the requirement as per the regulations of this institution and in our opinion meets the necessary standards for submission.

Dr. Daya Gupta

Professor, Former HOD
Department of Computer Engineering
Delhi Technological University, Delhi

Table of Contents

List of Figures

List of Tables

ABSTRACT

Agile software development has been attached much importance as a new software engineering methodology as it emphasizes on good communication between the developers, the rapid delivery of software, and change on demand. At early stages of software development, effort must be estimated to come up with a planned schedule and budget. From the Software Measurement point-of-view not every metrics and methods from conventional lifecycle models can be used without changes. In the recent past researchers have proposed different methodologies for cost estimation for software projects which use Agile Development methodology. In this thesis work we have proposed a new cost estimation model for Agile software development projects. In this model we apply Principal Component Analysis to reduce the dimensions of the attributes required and identify the key attributes which have maximum correlation to the development cost; and then use constraint solving approach to satisfy the criteria imposed by agile manifesto. We have extracted 12 factors (or components) for estimation of Development Cost and then constraint programming implementation using OZ is done so that the manifesto criteria are met. The proposed methodology is most suitable for agile projects as it uses constraint programming to explicitly check for satisfaction of agile manifestos. On comparison with other approaches under research we find that our model provides a low MMRE value i.e.50.63. Also the estimation error does not increase in high cost/complexity projects. Most Agile software Cost estimation processes rely on expert opinion or planning poker based cost estimation methods. However in case that is not available then our methodology can be used in case of unavailability of historical data or expert opinion. The extracted 12 factors can be used by the development team to estimate software development cost while still satisfying the conditions of agile manifesto. Hence we can safely say that the proposed cost estimation approach increases the precision and accuracy of estimates; and hence is better suited for the Agile Software Development Projects.

Chapter 1 INTRODUCTION

1.1 Introduction

Agile software development is the upcoming and new software engineering methodology. It stresses upon communication skills among the development team, the quickly submitting of software, and on-demand modification. Agile software development methodology incorporates practises like eXtreme Programming, Feature Driven Development, Scrum etc.try to decrease the cost of change and therewith reduce the overall development costs. It provides a way to check if a project is going in the correct direction in the development lifecycle. This is done by regular iteration of work, which presents a product increment potentially shippable to the client. These iterations are called sprints. Focus is on the repetition of small working cycles as well as the functional product which is their result. Agile software development methodology thus becomes "iterative".

Software estimation models help the development team in predicting of the probable amount of time and cost that will be required to complete the project development task.

1.2 Motivation for the work

In the initial stages of software development life cycle and planning stages, the development effort should be estimated so that the team can produce a plan, a time line called a schedule and the costing budget. In the current times software processes are constantly evolving. New and different technologies and applications are developed and are being currently used. In the current software industry scenario since software changes are arbitrary, requirement is of an evolving system for carrying out the estimation process, especially in agile environment.

Estimation the cost, size and duration (CSD) in the software development process has been a major point of the discussion and debate in all development methods; be it agile, waterfall, iterative or water fountain. The major issues being that of predictability and standardization. Organizations and development teams modify and personalise the estimation methods and development techniques so that they can be fitted over governance structures, culture and risk profiles. There is no one size fits for all solution.

The latest Release of **ISBSG (International Software Benchmarking Standard Group)** volume 12 has reported the fact that 9.8 % of the 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 as reported by the ISBSG vol.11 and vol.12 benchmark releases.

S.No.	Development Techniques	% projects
1	agile	9.8
2	datamodelling	32
3	event modelling	8.1
4	Joint application development	9.9
5	just in time, RUP	4.3
6	multifunction teams	9.7
7	process modelling	29.78
8	prototyping	16.2
9	rapid application development	5.4
10	reviews, walkthroughs,inspections	35.8
11	sandwich, hybrid	35
12	search based development	3.1
13	specific techniques described	15
14	traditional	71.3
15	uml, object modelling	17
16	waterfall	25.7

As reported by vol 12 release

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 %

As reported by vol 11 release

Fig 1. Different methodologies to develop software

We can see that there is an increase in the usage of agile development methodologies for software development. Hence it becomes all the more important to develop efficient estimation methodologies which can predict the development cost, effort and development time. So in order to support and enhance the use of agile methodologies need for some good estimation techniques arises which can provide accurate results.

Within this study we have investigated the domain of cost estimation practices for agile software development projects. For this we should also have knowledge of the extents and limitations or drawbacks of estimation tasks currently used in industry, in research and the challenges they put up. The following aspects are examined:

- conditions assumed by the projects that use agile activities process models
- Basic approaches and state of the art of the estimation in projects which currently use agile execution
- Applicability of the classic methods of the estimation and procedures, like code size or Function Points or COCOMO
- Experiences of the practitioners from the industrial and academic backgrounds while using of agile procedures

As a part of literature survey cost estimation activities for agile software development projects have been studied to gather knowledge about the possibilities and borders of cost estimation tasks and the new challenges.

1.3 Related work

From the Software Cost, Effort and Time Measurement point-of-view the metrics and methods from conventional lifecycle models cannot be conveniently used without changes.

J.M. Desharnais et all in [26] developed on the COSMIC (Common Software Measurement Consortium) method guidelines an estimation approach which works on COCOMO approach incorporating quality of documentation for functional analysis. It uses Cumulative Function Points (CFP) for estimation wherein each of the User Stories are defined by a single Function Point (FP) called User Story Point or USP. COSMIC has introduced the first official guideline in Agile software development methods for software sizing measurement based on function points.[27]

However the mapping between CFP and USP is subjective. It requires expert judgment and involves some degree of guesswork.

In another model given by Abrahamsson et all [16] predictors are extracted from current user stories and then used for next stories. This incorporates elements of analogy based estimates but works only in case the user stories are clearly written and well-structured.

In another research Zia et all [28] propose a SWOT analysis based method which uses influence of internal vs external factors and quantifies them. Factors such as team composition, process used, team dynamics, clarity of requirements etc are considered. In order to understand the factors that influence agility dimensions in a project, Lee and Xia[30] suggest a model which uses a trade-off relationship between response extensiveness and response efficiency of the team[30].

Asnawi et all in [20] used the Factor Analysis technique to identify 15 factors, by evaluating the responses they received in the survey. They explained contributions in several IT areas such as process/governance, quality assurance, iterative and incremental development and team communication. However these were not fully related to the aspects of projects which impact project performance such as cost, quality, deadlines and scope.

In the literature survey it was observed that recent researches are using PCA based models for software cost estimation for traditional software development.

Tosun et al. in [32] proposed feature ordering in terms of PCA based factor-importance and provided heuristics for it. They ordered the features according to absolute values of the elements of the eigenvector of the first principal component only.

and the features were ordered .

In [34] the researchers J Weng, Shixian Li, Linyang Tang demonstrated how PCA based models can provide significant improvement in reliability and accuracy of effort prediction over Traditional analogy based models. They compared the performance of PCA-based feature extraction with analogy based methods on three public datasets namely COCOMO, NASA and Desharnais. It was found that their WPCAA model outperforms the traditional analogy based models in terms of MMRE and PRED(25).

Drawing inspiration from this we explored the use of PCA based cost estimation model in the Agile Software Development environment.

1.4 Research Problem

In the recent past researchers have proposed different methodologies for cost estimation for software projects which use Agile Development methodology. These methods use large

number of project characteristics such as story points, story size, factors related to team dynamics, process model used, factors related to management, communication skills in the team, quality and clarity of requirements etc.. to estimate development cost.

In this work we apply Principal Component Analysis so that we can significantly reduce the dimensions of the attributes required; and identify the key attributes which have maximum correlation to the development cost. This might improve the cost estimation process. Also the constraints of agile manifesto are considered which may increase the accuracy of cost estimation.

The Research Problem can be thus be stated as follows:

To propose a robust cost estimation model for agile software projects, which extracts key factors using Principal Component Analysis and estimates the development cost using these extracted factors and satisfies the constraints imposed by the agile manifesto.

1.5 Scope of the work

This thesis reports on the results obtained by exploratory factor analysis carried out on Agile development data of various industry projects. The proposed cost estimation model uses a PCA based approach to derive the correlation between various project attributes and the cost of development. A constraint programming based approach is used to estimate the development cost while incorporating rules from the agile manifesto.

The scope of the thesis is presented as follows:

(i) We do exploratory factor analysis of the agile development data. Using the Principal Components Analysis technique, we extract factors called as Principle components (PC) and their correlation matrix. These PCs are the factors which have most affect on the development cost and they account for most variation in the agile development data.

(ii) Using constraint programming implementation using OZ we impose the agile manifesto restrictions on the extracted factors. The weights as obtained by PCA along with rules pertaining to agile manifestos are taken as the inputs and estimated development cost is output.

(iii) The proposed model is validated by comparison with the existing cost estimation models in terms of mean magnitude of relative error (MMRE) to show improved accuracy of estimates.

1.6 Organisation of thesis

The subsequent chapters of the thesis are organised as follows:

Chapter 2 briefly explains the Agile development environment and the different types of Agile development methodologies.

Chapter 3 discusses the software cost estimation models as used in traditional software development, such as LOC model, Regression models, COCOMO, Bayesian model etc.

Chapter 4 introduces the cost estimation in agile development environments. This chapter explains the differences in estimation in agile environment and traditional software development environment, the challenges faced and planning poker estimation technique as used in industry.

Chapter 5 presents several estimation approaches in research.

Chapter 6 presents the proposed approach for software cost estimation and explains PCA and the constraint programming approach.

Chapter 7 presents the implementation details of the proposed model on the given agile data. It also presents the results of implementation and analyses the results. In this chapter we have also shown a comparison of the proposed approach with previous estimation approaches and analyses the results.

And then we conclude the thesis.

Chapter 2 Agile Software Development

2.1 Introduction

Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen tight interactions throughout the development cycle.

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.

2.2 Agile development Environment

Agile development methods promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

- **Iterative, incremental and evolutionary**

  Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (time-boxes) that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration. Multiple iterations might be required to release a product or new features. Demonstration will minimizes overall risk in the development of project and

hence will allows the project for adapting to changes rapidly. Each iteration will not increase enough functionality to the software that it can be released to the market or client-release. However but the goal was to make available a small version of the product having some or all the functionality but with minimal bugs at the end of each iteration. Many such iterations are to be required to release a product or new features.

- **Extensive communication among team members**

  In agile teams the stakeholder or the customer appoints a representative like a Product Owner for SCRUM. He will be acting on behalf of the customer/client and has the decision making power. He is the person who will approve changes done by the team. He will also foresee the team's work so as to a make sure that they are on the correct track of development. If the team has any doubts or queries in the ,middle of the iteration he is the person responsible for correct answers. His interpretation of client requirements are considered of utmost importance.

  In 2002 Alistair Cockburn coined the term "information radiator". It is used to inform the whole team and other stakeholder about the status of the project and the direction in which the product is headed. It is normally in the form of a big physical display located prominently in an office, where every member can see it.

- **Very short feedback loop and adaptation cycle**

  A common characteristic of agile development are daily status meetings or "stand-ups", e.g. *Daily Scrum (Meeting)*. In a brief session, team members report to each other what they did the previous day, what they intend to do today, and what their roadblocks are.

- **Quality focus**

  Specific tools and techniques, such as continuous integration, automated unit testing, pair programming, test-driven development, design patterns, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance project agility.

2.3 How is Agile Different from Traditional Software Development

Barry Boehm described agile methods as "an outgrowth of rapid prototyping and rapid development experience as well as the resurgence of a philosophy that programming is a craft

rather than an industrial process" [4]

Table 1. Differences between traditional approach and agile approach to software development[1][4][6][8][13][14]

TRADITIONAL APPROACH	AGILE APPROACH
Deliberate and formal, linear ordering of steps, rule-driven	Emergent, iterative and exploratory, beyond formal rules.
Optimization is the goal	Adaption, flexibility, responsiveness is the goal
In this type the environment is taken as stable and predictable.	In this type the environment is taken as turbulent and difficult to predict
Sequential and synchronous process	Concurrent and asynchronous process
It is work centered process because people will change according to different phases	It is people centered process, as the same team is developing throughout.
Project lifecycle is guided by tasks or activities.	Project lifecycle is guided by product features.
Documentation is substantial.	Documentation is minimal.
Developers do waiting until the architecture is ready.	The whole team is working at the same time on the same iteration
Too slow to provide fixes to user	Provide quick responds to user feedback
Change requirements is difficult in later stages of the project	Can respond to customer requests and changes easier
More time is spent on design so the product will be more maintainable. The "what ifs" arise earlier	There is no time for the what ifs
No communication within the team, novices stay in their rooms and try to understand things	High level of communication and interaction, reading groups, meetings
Restricted access to architecture	The whole team influences and understands the architecture. Everybody will be able to do a design presentation
Documents and review meetings are needed to solve an issue	5 minutes discussion may solve the problem
Everything is up front, everything is big before you start	The focus is on whether customer requirements are met in the current iteration

2.4 Agile Manifesto

The four agile manifestoes[1] that are a concise summary of agile values in software development are given below:

1. **Interactions among team members are given precedence over formal processes and tools.**

   This value factor essentially concentrates on the quality, efficiency and skills of the development team members. The criteria like communication skills, managerial skills and computer science skills play an important role than the process quality and the sophistication of tools used.

2. **Working software is more important than comprehensive documentation.**

   Agile team performs iterations and may not devote much time and effort for the purpose of documentation. It develops product for its functional and non-functional requirements at regular intervals and believes in delivering the working model for each increment rather than elaborate documentation.

3. **Customer collaboration is preferred over contract negotiation.**

   In-place of giving important to the formalisation of contract, agile software development will concentrates on its clients beforehand. Customers are very much involved in the software cost and schedule estimation tasks so much so that if the development team has facing difficulty in understanding any user story they can immediately have discussion with the customer and try to break the story down further [5].

4. **Quick and appropriate response to change in requirements is given precedence over a fixed plan.**

   Agile methods welcome changing requirements, even late in development stage. They accept changes in the requirement specification and allow a flexible software development without being restricted to a fixed plan. The usual agile approach is to fixing of the time schedule and development cost, and then scope of the project is allowed to vary in a control-specified manner.

2.5 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.

Some of the characteristics of the agile processes are as follows:

- The software versions are released quickly one after another as the iteration size and the release cycles are short,
- The complex functionality is designed using an easy to understand architecture,
- The clients and developers and management team, all take collective ownership over the product
- High coding standards are followed so as to produce a very sophisticated code; Refactoring and re-use of code
- continuous testing of the code just as they are developed; comprehensive regression tests and acceptance tests by the clients
- Continuous integration so that the product can be viewed a whole

2.6 Types of Agile methodologies

2.6.1 Extreme Programming

Extreme Programming (XP) is the most commonly used agile software development method. It is also seldom used a reference when we talk about the general characteristics of Agile software development methodologies.

According to XP principles the software development projects need to focus more on the people involved rather than the documents, processes and tools. XP provides a set of practices, values and principles [29]:

- Values : communication, simplicity, feedback, courage
- Principles : incremental changes, honest measuring
- Practices : pair programming, short version cycles

These are derived from the industry experienced best practices. It has been documented and experienced that XP helps the teams in software development projects. To fulfil some of these completely varying project characteristics, the agile software development model establishes an XP product life cycle just like traditional life cycle models such as waterfall-model, or spiral model [28].

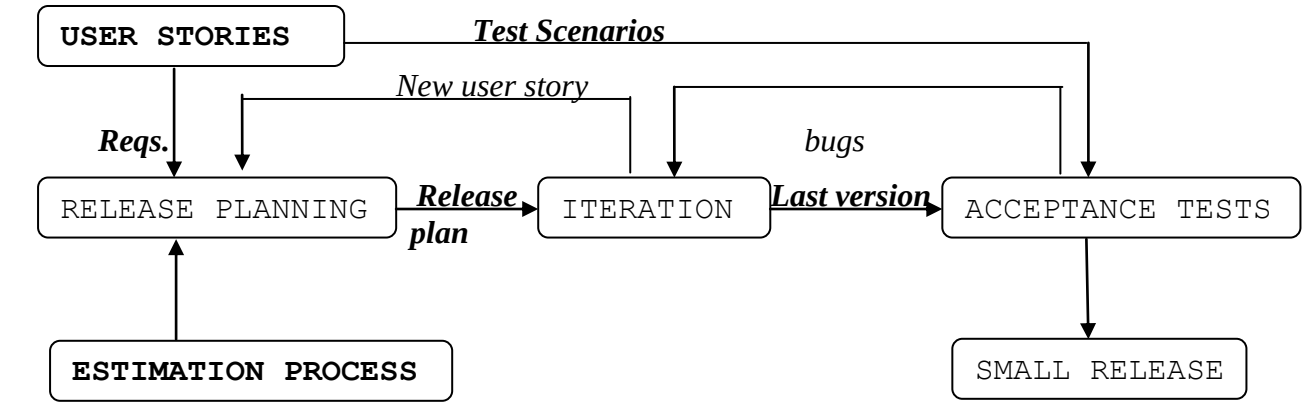

Fig 2. General Way of Developing Software through XP

The figure 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

For example let us consider software project for an online mood-board application. A mood-board is where an artist can place various different media- pictures, audio-video, fonts, text, drawing etc as an inspiration for his project.

The requirements are given by the clients to a set of developers. They discuss these requirements with the remaining of the team. The scope and extend of each requirement is discussed. The developers divide the feature development work among themselves. In XP the focus is on the creation of a working program for the software. The program code is given utmost importance and the whole development process is focussed on the programming aspect.

So the requirements of the online mood-board application could be given such as- to be able to select an image to add to the board, to be able to perform basic editing on that image, to be able to add fonts to the board, to be able to save and share the creation etc.
The xp processes say that the whole application's coding should begin at once. All the developers start the coding work on their assigned features. In each iteration either a part or whole of the feature will be developed. If a feature is not fully developed in the iteration then in the next iteration remaining work is done. This way in each iteration the original code or the originally developed application is modified upon.

2.6.2 Agile Modelling (AM)

The software modelling is done before its development is started. It gives the developers a blue-print of the program to be developed. It also helps the developers to identify any issues or problems they are likely to come across while development. If modelling of the program is done before development then these issues can be discussed and mitigation strategies can be made before-hand.
Agile Modelling is the term for modelling in agile software development. Agile modelling (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.
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 also include

- Knowing why the Agile modelling is being done
- Presence of other effective models also;
- Importance is given to the content more than its look and representation;
- Extensive communication among the stakeholders and all those involved in the development process;
- focusing on the work/ product quality [2].

The practices of AM have some commonalities with those of XP, too. An agile modeller 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 [11].

2.6.3 SCRUM

SCRUM methodology was initiated by Ken Swaber in 1995. It was practiced before the announcement of Agile Manifesto. 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.

Its principle lies in the fact that small teams working cross functionally produce good results. Scrum is more revenue centric with attention on improving revenue and quality of the software. Since being lightweight it can adapt to changing requirements and releases the software in small release cycles called sprints.

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 [5].

For SCRUM, three main roles are defined.

- The first role is the product owner. His task is to make sure that the business aspect of the software project is taken care of.
- The second role is the SCRUM team who take care of the development work aspect. These comprise of developers, testers, and other technically skilled people.

· SCRUM master, the third role, is responsible for keeping the team focused on the specific tasks [4][3].

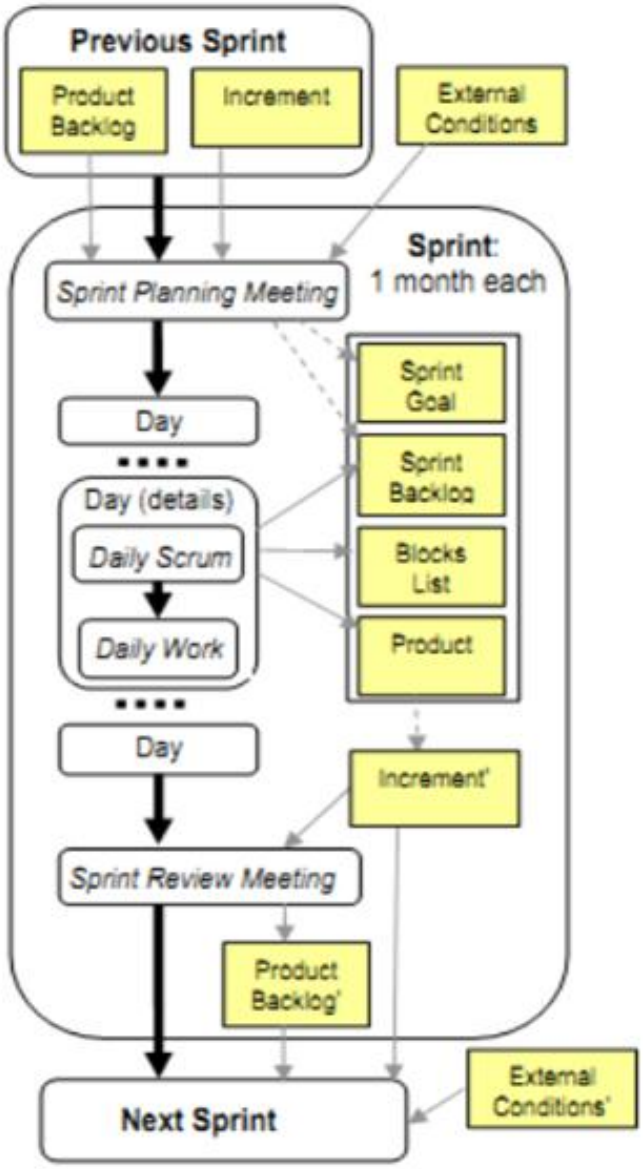

Fig 3. Product development in Sprints

The process of development using SCRUM is done in phases. The development tasks are divided into stages. In each phase, some of the functionality of the product is fully developed, tested, and made ready to go to production. Only after completing one phase does the team move to a new phase.

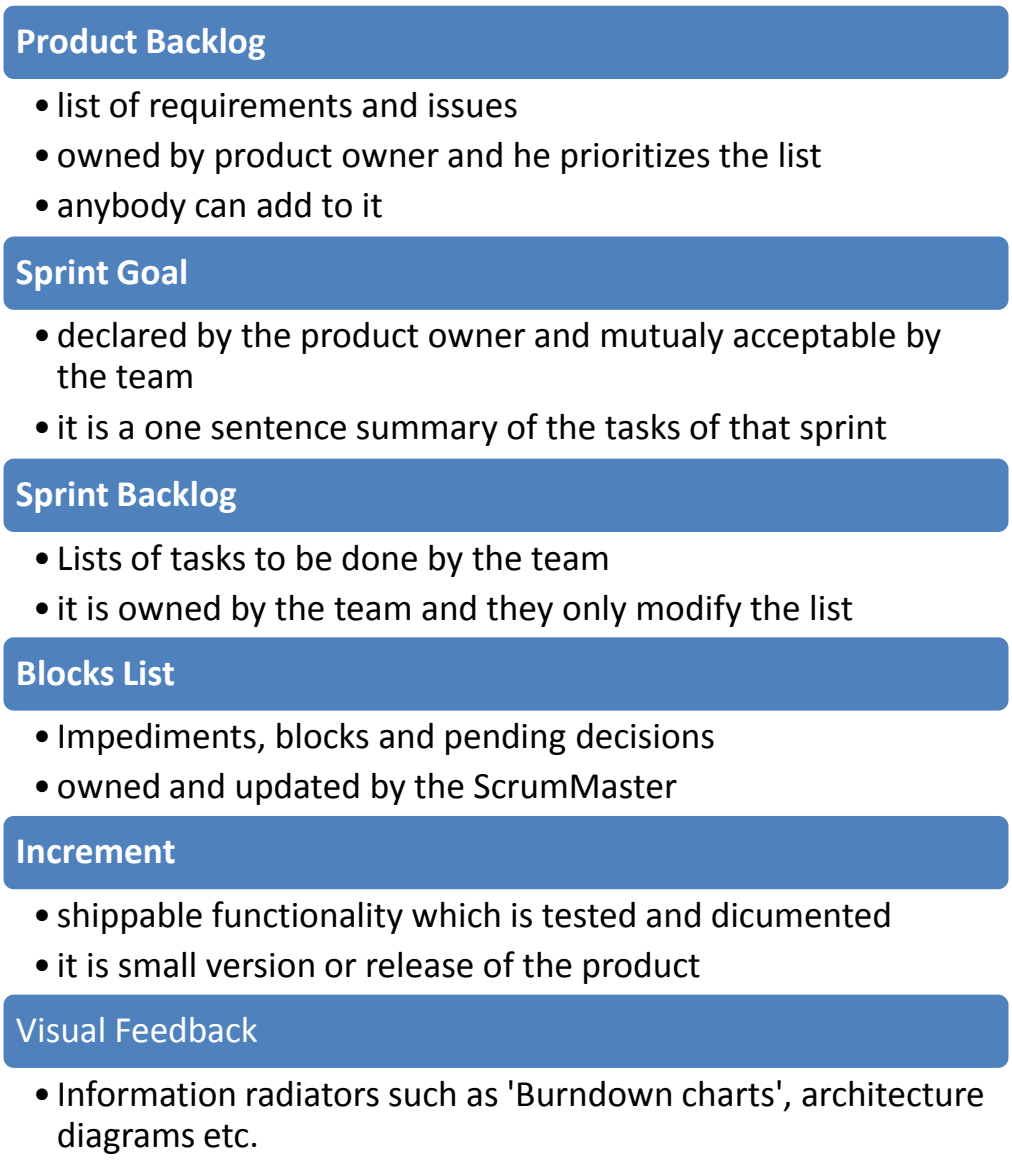

Fig 4. Key Artifacts of Scrum

Let us consider the previous example of an agile team of developers who have to develop a software for an online mood-board application. A mood-board is where an artist can place various different media- pictures, audio-video, fonts, text, drawing etc as an inspiration for his project. The client gives requirements which are converted into user stories.

The user stories are written in the format:

"As a <role> i should be able to do <desired feature>"

These user stories are then divided so that they can be implemented in different small iterations. Consider some of the user stories such as-

- As an artist i should be able to add images from my computer to the mood-board.

- As an artist i should be able to search for images from Google image search and add it to the mood-board.
- As an artist i should be able to select the size and shape of drawing tool
- As an artist i should be able to add background colour and texture to the mood-board

Now these stories are considered one by one by the product team. Developers are assigned user stories and they become the owner of that feature.

Now the coder will provide an estimate of how much effort or time will be required for developing that feature. The testing personnel will estimate how much time will be required for testing it. The project management adds one or two days for legal checks and internationalisation checks. And the total project effort and time schedule estimate is produced.

When the iteration starts each day a small part of the user story is implemented by the developer. That part is also tested simultaneously by unit tests.

Every day the team gathers for a scrum meeting. This meeting is supervised by a scrum master who is usually the program manager. In the scrum meeting the team discusses :

- The work completed till now
- The work in progress
- Problems and issues faced by each individual
- Any requirements from other teams

These issues are discussed daily by the team so that all the stakeholders are completely aware of the project progress.

The user stories which are completed till the scheduled time become features for the release of that iteration. The remaining user stories are added to product backlog i.e. to be completed in the next iteration. The acceptance tests are carried out by the clients and issues or bugs are logged. If there is any change in the requirements by the client then the product owner adds it to the product backlog. It is taken up as a new feature in the subsequent cycles.

The features approved by the clients are then released in the form of a new version of the software.

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 [5]. 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.

2.6.4 The Feature-Driven Development (FDD)

This approach focuses on the software features of the system. They are the main driver of the entire development process. It differs significantly from the other agile processes because they put a strong emphasis on planning and upfront design. As shown in Figure 3, the first step of the FDD process is to build a detailed model of the system, which captures all the stakeholders' assumptions and requirements. Once the domain model is built, the team members print a list of the features of the system.

Each feature has to be developed in a few hours or days, but no longer than 2 weeks. Using FDD, development teams are divided according to design and to implement a particular feature. The development work is performed in parallel on all the features. Each team is headed by a feature owner, who is responsible for the code segment that implements those features. This is contrasted with the XP approach where the ownership of the code belongs to the whole development team and not to a specific member.[13]

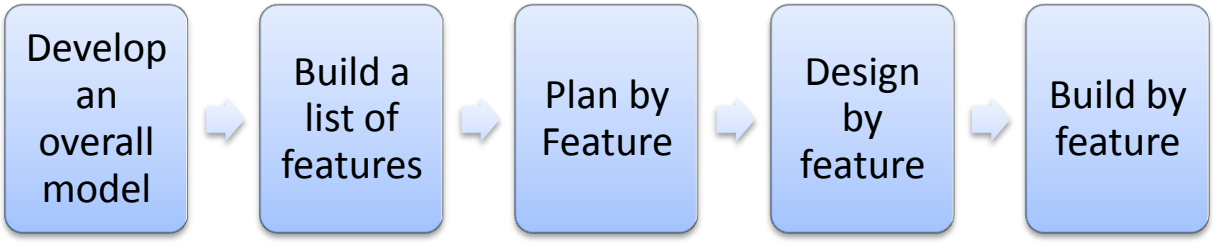

Fig. 5 Simple Steps of FDD[6]

Let us consider our previous example of the online mood-board application. Initially the features of the software will be considered. These are- adding images to the board, drawing on the board, adding a background, adding audio-video content, saving and editing the board at a later stage, sharing the creation etc.

In each iteration some of the features will be considered. Consider that the feature 'adding images to the board' is one of features selected in the current iteration. This feature will be owned by the whole team as opposed to a single developer as in scrum or XP. The work on all the features selected in the current iteration will be started. The focus on development is to complete as many features as possible in the current iteration.

If one developer completes the work on one feature then they can help the other members of the team in the completion of the in-completed features rather than starting on an new feature.

The FDD process enforces rigorous guidelines in order to find defects in the system. It also enforces coding standards and encourages regular builds on a daily or weekly basis in order to add freshly designed features to the base system. Since the features are developed in parallel, it is mandatory to have a configuration management system that allows calmly integrating of the changes that are made to the system. In FDD approach there is a tracking and papering mechanism that is used to show the project status based on the number of features that have been implemented as well as the overall progress of the design, coding, and testing activities. Each feature is scored using a value ranging between 0 (for a feature that has not yet been worked on) and 1 (a completed feature) and anything in between refers to a feature in progress.[13]

2.7 Why is Agile Development Necessary

In 1970, Dr. Winston Royce presented a paper entitled "Managing the Development of Large Software Systems," which presented the short-comings of sequential development process used for software projects. He said that software should not be developed like an automobile on an assembly line, in which each piece is added in sequential phases. In this type of sequential development in phases, one phase of the project has to be done and then only the next phase can begin. Dr. Royce spoke against such phase based approach in which developers will initially be gathering all of a project's requirements, then completely designing of all of the architecture and design, then writing all of the code, and so on. Royce's paper specifically mentioned the drawbacks of the approach where there is not

enough communication among the smaller teams that are working on a particular phase of work.

In waterfall model, development teams get a single chance only for each aspect of a project development. They should produce correct result in that chance only. In an agile modeling, every feature of development — like requirements, design, etc. — is again and again done throughout the lifecycle. At the end of each iteration if there is any mistake or changes need to be done, it is easy to do so.

By using agile development methodology we can re-evaluate the direction of a project at any point during the development lifecycle. This is done by developing the project in small iterations, known as sprints. At the end of each sprint the teams presents a potentially shippable product increment to the clients. Hence, agile methodology can also be described as "iterative" and "incremental".

.

2.8 Difficulties faced during implementation of agile methods

2.8.1 Fear of Exposure of Skill-Deficiency

In a review of 17 companies, it was found that the development team was scared that the agile process can highlight the gaps in their skills and expose their deficiencies[11]. So, they felt a pressure at all times while using agile methodologies.

To mitigate this problem, the developers need an atmosphere in which they feel the safety to project their weaknesses. They should be able to document any fears, issues or concerns due to which they didn't feel comfortable in an open forum.

2.8.2 Broader Skill Sets for Developers

Generally in software companies using agile development methodology the management requires personnel to exhibit a wide range of skill set rather than specialisation in only one area like program writing using a particular language or build deployment only [11].

To address this problem, organization goals and HR policies and expectations should be realistic. They must strive to provide their employees a well-balanced team with members becoming "masters of all" or "masters of none." The ideal situation will be when the developers have broad knowledge of the stages and features of software development however they are experts in certain areas.

2.8.3 Interpersonal Interaction among team members

Agile practices encourage collaborations, among developers and with the other stakeholders. This leads to meetings, retrospectives etc. which require the social interaction. For this the team members hone their inter-personal, communication, and presentation skills. In most of the cases management prefers constant face-to-face communication, as they could see the benefits of increased the degree of communication in agile environment. However there are people who were technically very talented but had weak communication and presentation skills[11].

This challenge can be met by providing social-skills training to the development team so that they can be more comfortable in such social work settings.

2.8.4 Understanding Agile Principles

Not all projects are suitable for application of agile values and principles. Sometimes due to irregular combination of staff personality, incorrect management style, company policy or any other factors the projects development teams were forced to implement agile methods. However it was only “on paper,” and they could not achieve agility’s ultimate goals.

Chapter 3 Cost Estimation in Traditional System Development

3.1 Introduction

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

The main factors that are typically estimated at the beginning of an IS development project are: cost, size, schedule, people resources, quality, effort, resources, maintenance costs, and complexity. Estimates are produced and used for a variety of purposes and a study [7] revealed the most common uses. These are: to schedule projects, to select proposed projects for implementation, to quote the charges to users for projects, to staff projects, to audit project success, to control or monitor project implementation, to evaluate project estimators, and to evaluate project developers.

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.

3.2 Cost Estimation Techniques

Cost estimation tools, or model-based estimation techniques use data collected from past projects combined with mathematical formulae to estimate project cost. They usually require factors such as the system size as inputs into the model.

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.

Lines of Code (LOC) is most commonly used as a size measure. The major software cost and schedule estimation techniques are given in the following sections. These are classified as regression-based models, learning-oriented models, expert based approaches and composite-Bayesian methods.

3.3 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.

3.4 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.

3.5 Regression models

Most of the software estimation models that are available are based on some form of regression technique. Regression models have a mathematical foundation and are constructed by collecting data on completed projects and developing regression equations that characterise the relationships among the different variables. Estimates are made by substituting the new project parameters into the mathematical model with the use of a large data set. Statistical regression models estimate software development effort as the dependent variable.

Regression models however can be difficult to use in some cases, in particular if they do not satisfy a number of conditions that can either enhance or halt successful use.[18]

These 4 conditions are based on experience from the use of regression-based models. They are: availability of a large dataset, no missing data items, no outliers, and the predictor variables are not correlated. The collection of approaches that fall under the heading of regression-models include ordinary least-squares regression, classification and regression trees, stepwise analysis of variance for unbalanced data sets, combinations of CART with OLS regression and analogy, multiple linear regression, and stepwise regression.

3.6 Learning-oriented models

These attempts to automate the estimation process by building computerised models that can learn from previous estimation experience. These models do not rely on assumptions and are capable of learning incrementally as new data are provided over time. Learning-oriented models cover a wide area and include techniques such as artificial intelligence approaches, artificial neural networks, case-based reasoning, decision-tree learning, and machine learning models, knowledge acquisition, fuzzy logic models and rule induction. Cost estimation tools such as COCOMO, learning oriented and regression-based models; expert knowledge; and composite-Bayesian methods. [21] The main model-based techniques include COCOMO, SLIM, RCA PRICE-S, SEER-SEM, and ESTIMACS. These estimation models produce an estimate of the cost, effort or duration of a project based on factors such as the size and desired functionality of the system.

3.6.1 COCOMO Model

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current as well as future project characteristics.

The COCOMO (COnstructive COst MOdel) cost and schedule estimation model was originally published in [Boehm 1981]. But COCOMO '81 along with its 1987 Ada update experienced difficulties in estimating the costs of software developed to new life-cycle processes and capabilities. The COCOMO II research effort was started in 1994 at USC to address the issues on non-sequential and rapid development process models, reengineering, reuse driven approaches, object oriented approaches, etc.

In 1995 COCOMO II was developed and finally published in 2000 in the book Software Cost Estimation with COCOMO II. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software development technology moved from mainframe and overnight batch processing to desktop development, code reusability, and the use of off-the-shelf software components.

COCOMO model proposes following **product classes**:

(i) Application Programs- Data processing and scientific programs

(ii) Utility Programs- Compilers, linkers, editors, etc.

(iii) System Programs- Operating systems and real-time system programs, etc.

Another Classification is on the basis of the team composition and size.

(i) Organic- Relatively small groups working to develop well-understood applications.

(ii) Semidetached- Project team is a mixture of experienced and inexperienced staff.

(iii) Embedded- The project team is highly specialised and experienced as the software is strongly coupled to complex hardware, or real-time systems.

COCOMO model comprises of a hierarchy of three forms. The first level, Basic COCOMO is used for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes. Intermediate

COCOMO takes into account these Cost Drivers and Detailed COCOMO accounts for the influence of individual project phases.

3.6.1.1 Basic COCOMO

Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC)

COCOMO applies to three classes of software projects:

- Organic projects - "small" teams with "good" experience working with "less than rigid" requirements
- Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements
- Embedded projects - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects- hardware, software, operational,

The basic COCOMO equations are given as follows:

$$\boldsymbol{E} = p \times KLOC^{q}$$

$$\boldsymbol{D} = r \times KLOC^{s}$$

$$\boldsymbol{P} = {}^{E}/_{D}$$

where, KLOC is the number of delivered lines of code written for project expressed in thousands. E = Development Effort Required in units of person-months. D = Development Time in units of months and P = number of People required in the development team.

The estimated effort when plotted against the size of the project given the following graph:

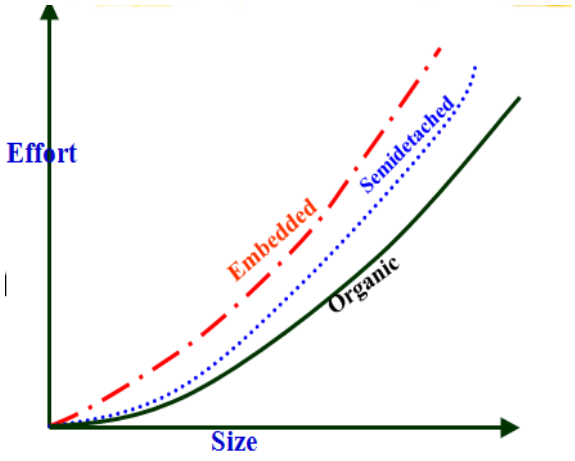

Fig 6. Estimated effort v/s software size

Development time is a sub linear function of product size.

When product size increases two times, development time does not double.

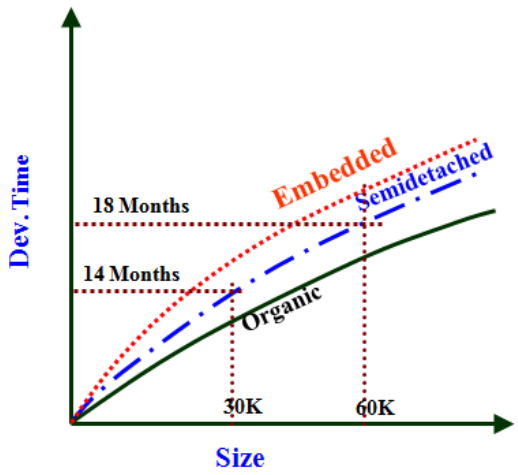

Fig 7. Development Time v/s software size

3.6.1.2 COCOMO II

COCOMO-II was initially published in the Annals of Software Engineering in 1995 [Boehm et al. 1995]. The model has three submodels, Applications Composition, Early Design and Post-Architecture, which can be combined in various ways to deal with the current and likely future software practices marketplace.

COCOMOII computes software development effort as function of program size and a set of "cost drivers".

- **Product**: Inherent complexity of the product, reliability requirements etc.
- **Computer**: Execution time, storage requirements, etc.
- **Personnel**: Experience of personnel, etc.
- **Development Environment**: Sophistication of the tools used for software development.

Each of the attributes is rated on a six-point scale ranging from "very low" to "extra high" (in importance or value). An effort multiplier is applied to this rating. The product of all effort multipliers is called the ***effort adjustment factor (EAF).***

The Intermediate Cocomo formula:

$$\boldsymbol{E} = p' \times KLOC^{q'} \times EAF$$

where E is the estimated development effort in units of person-months, KLOC is the software size in terms of number of thousands of lines of code for the project.

3.7 Expert Judgement

The most widely used estimation approach was found to be "comparison to similar, past projects based on personal memory". These expertise based approaches are useful when no quantified, empirical data is available [15], and they provide a practical, low-cost and highly useful process. Estimation by analogy in its most basic form involves examining past projects and using the information retrieved as a guide estimate for the proposed project. The Checkpoint method is an example of an analogy-based approach to software estimation. Rules of thumb can be derived from actual project data or a formalisation of expert opinion, either way they must make use of some form of project data or information. Rules of thumb can be used to estimate productivity, quality or size. Expert judgement relies on the accumulated experiences of teams of experts in order to come up with project estimates. This technique is used where the estimation process is primarily based on "nonexplicit, non-recoverable reasoning processes", or perception and intuition highlight the weaknesses of using any human memory-based techniques, because past projects can be forgotten, details confused and important factors accidentally ignored. [14]

The very nature of expert judgement means that deriving an estimate is not a repeatable process; however reports have proven it to be the dominant strategy in software development. The Delphi technique and work breakdown structure (WBS) fall under the heading of expert judgement techniques. These help to reduce the likelihood of errors occurring in the estimation process. [15] Other techniques in the expertise-based category include top-down and bottom-up estimation reasoning by analogy, formal reasoning by analogy, informal reasoning by analogy, and rules of thumb. A study revealed that the most widely used estimation approach was "comparison to similar, past projects based on personal memory". Expertise based approaches come under much criticism for their reliance on human memory and the lack of repeatability of such memory-based approaches, however reports have proven it to be the dominant strategy in IS development estimation [13].

3.8 Bayesian approach

The Bayesian approach is a semi-formal estimation process that combines the strengths of expertise based methods and regression-based methods. Bayesian analysis allows for the fact that the data required for use in most estimation techniques is typically of poor quality or incomplete. Expert judgement is incorporated in this approach to handle the missing data and provide a more robust estimation process. Bayesian analysis has been used in many scientific disciplines and was used in the development of the COCOMO II model . Cost Estimation, Benchmarking and Risk Analysis (COBRA) is an example of a composite estimation model. It requires expert knowledge and a relatively small amount of quantitative data gathered from past projects in order to produce estimates of a project's cost and development effort, and also the quantitative risks associated with the project.

3.9 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 progress 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 a 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.

3.10 Causes of Inaccuracy in Estimation Process in Software Development

There is always an inherent error associated with any form of estimation. This is primarily because "an estimate is a probabilistic assessment of a future condition" and therefore accuracy is expected to be low in tany kind of estimation process. The causes of inaccurate estimates in development projects for Information Systems applications (IS) can be grouped into five categories by Lederer and Prasad [7], namely methodology, politics, user communication and management control, uncertainty.

3.10.1 Methodology refers to the estimation process adopted and includes the steps undertaken to produce the estimate and also the means of examining and reviewing the estimates relating to past projects. Over reliance on intuition and personal memory is a concern for project members trying to increase estimation accuracy [7]. Estimation inaccuracy can also be caused from a lack of procedures and policies on how to deal with failures and avoid repeating mistakes by learning from past experiences. [8]

3.10.2. Political forces at work within a project or company can often drive estimation inaccuracy. This is usually in the form of managerial pressure to stay within or meet the estimate. The estimation process can be impacted negatively by these pressures resulting in time or cost constraints. When estimates are produced simply in order to satisfy managers or customers it will inevitably lead to inaccuracy [7]. According to Jorgensen and Moløkken [10], it should be recognised that estimation is typically fraught with "tug of wars" and "political games", and thus high estimation accuracy is usually not the only goal of the actors involved. Moløkken and Jørgensen [10] suggest that software managers may over-report causes of inaccuracy that lie outside their responsibility, such as customer-related causes. Project managers therefore have to be aware of the implications that political factors can have on IS development estimation.

.

3.10.3. User communication refers to the factors relating to the customers and their changing requirements throughout a system's life-cycle. This is usually the most prominent factor in causing project estimates to be inaccurate [10]. Incomplete or unclear requirements specification at the beginning of an IS development project is typically due to the fact that customers have to deter in what their requirements are and they are usually unaware what the current state of the art is, or what the competition is doing. This leads to difficulty in producing a complete set of requirements and thus estimation inaccuracy is inevitable[9]

3.10.4. Management control includes management reviews, and comparison between estimates and actuals. When management fails to participate in the preparation of the estimate, and does not monitor the accuracy of the estimate, this is believed to contribute to the estimate being inaccurate. Inaccuracy also occurs when management does not refer to the estimate when conducting performance reviews of estimators and other project personnel [7]

3.10.5. Uncertainty: Due to lack of clear understanding of requirements and allowing change in requirements, there is a lot of uncertainty in the requirements.

However the uncertainty extends further than requirements into factors such as those that can be purely technical (whether specific coding languages can do the job) to the complexities of the real world. Moving back toward the beginning of the project where most estimation activities take place, one simple truth becomes apparent, knowledge dispels uncertainty. We need to gather better knowledge whether by leveraging history, mathematical algorithms and/or project specific information to make better estimates. Integrating agile techniques for knowledge capture in projects are tools for reducing uncertainty. Techniques to use include incorporating a user or user proxy on the team, focusing on short pre-defined time horizons, implementing processes that foster communication and periodic re-planning. [8]

Chapter 4 Cost Estimation for Agile Development Methods

4.1 Introduction

Agile methods "welcome changing requirements, even late in development" [1].for the purpose of cost, schedule and effort estimation, the requirements are semi-finalised at the start of each iteration so developers can carry out the planning and estimation process safely knowing that the scope for the iteration has been finalised and unchanging. Simplifying the early estimation effort is done during the initial release planning sessions where the estimates produced at this early stage for the entire project are typically at a high level. Carrying out frequent estimation tasks during and before planning at the beginning of every iteration, leads to progressively more accuracy in the estimation process by the developers.

4.2 Agile Estimation

The agile manifesto [1] recommends that cost-sensitive projects should use serial development where possible, while projects sensitive to shifting requirements benefit more from concurrent development. It then goes on to state that agile project teams almost always use concurrent development, which implies that agile methods are not wholly suitable for projects that are cost-sensitive, although realistically it is rare to find a project that is not.

In terms of the agile development, the estimation process is an iterative one whereby the user stories in eXP represent pieces of functionality to be estimated and this is done every two weeks. An overall expected time for each of these stories is estimated by the developers, and the customers then prioritise the stories based on these initial estimates and on the business value of each one .

The techniques used to estimate agile development projects have typically been expertise-based, where the developers look to past projects or iterations, and draw on their own experiences to produce estimates for the stories.

A study [6] claimed that none of the companies had used COCOMO and that 40% used function points estimation on their agile projects. These results however are based on only few companies and so do not represent generalise-able data, n\although from the available literature there does seem to be an inclination toward the reliance on expertise-based estimation approaches. [13]

The popular and most common approach for effort estimation in agile methods is subjective estimation [22]. Although this approach is simple and easy to apply, estimates are highly biased. In some agile teams, effort estimation is based on their previous iteration actual effort and hence effort estimation is useful only for remained user stories. In addition application of planning poker is one the most popular practices for many agile teams in planning and predicting effort before starting each iteration [23]

4.3 Traditional Estimation Methods vs. Agile Estimation Methods

Traditional Estimation Methods (TEMs) and Agile Estimation Methods (AEMs) possess some differences and equivalence in used terms based on their computing practices. TEMs consist of methods based on both algorithmic and non algorithmic approaches whereas AEMs follow mainly non-algorithmic approach for CSD estimation of project. TEMs assume that the requirements are well formalized before beginning of CSD estimation. On other hand, AEMs deal with a small set of requirements (which may grow organically in later stage of software development).

Furthermore, TEMs derive the estimation for whole software but AEMs compute Cost for working software in estimation process. On contrary to AEMs, TEMs consider various factors affecting the CSD during the estimation process.

On other hand, TEMs and AEMs use the same quantitative information in terms of productivity and team size for CSD estimation of the project. The equivalence of factors/ terms that are used in both TEMs and AEMs is discussed as follows:

Size and Complexity: Size of the project is defined in terms of OPs and FPs in TEMs and computed through counts of various inputs, logical files, external file, screens etc. AEMs use story points to define the size of a story and sum of all story points is defined as size of the project. Size and complexity of project are major factors in cost and duration estimation and are computed in the form of OPs, FPs in TEMs (COCOM II, FP based estimation) and story points in AEMs. Thus, story points, FPs and OPs are equivalent to each other. However, OPs

and FPs in TEMs use mathematical formula for computation and story points are estimated through heuristic approaches.

Efforts: TEMs introduce the term person per month for computing efforts spent on project. Similarly, AEMs uses the term elapsed time which is defined as time spent on project by a person. Person per month is total hours spent by a person on the development of the project same elapsed time as in AEMs. AEMs also introduce ideal time for estimation that refers to time required on a project without any interruption.

Productivity: Productivity rate in TEMs is equivalent to velocity in AEMs. Productivity rate is computed through function of person per month and OPs. Velocity in AEMs is number of story points completed per iteration. Velocity/ productivity rate represents team competence on project and it is used to estimate time and cost of the project.

Table 2. Comparison of traditional estimation methods and agile estimation methods

	TEMs		AEMs
	COCOMO II	**FUNCTION POINT**	
Size and complexity	Object points	Function points	Story points
Efforts	Man-month	Man-month	Elapsed time/ Ideal time
Productivity	PROD=NOPs/ person per month	Calculated in terms of FP	Velocity= Number of story points per iteration

4.4 PLANING 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. The participants are all the programmers, testers, analysts etc. working on that project.

The Steps of planning poker estimation are:

1. Each estimator is given a deck of cards with a valid estimate written on it in numerical form. The numbers may for example be from Fibonacci Series 0, 1, 2, 3, 5, 8, 13 and 21. The numerical estimates are usually written in non-linear form. These non-linear sequences correctly 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 [24].
2. A moderator is assigned for the estimation process. The moderator is usually the product owner (customer) or an analyst. He reads the user story description. The

stories are discussed in detail and any queries regarding the scope and complexity of the story is answered by the product owner.

3. Each estimator will decide on an estimate for the cost or effort required for that story to be completed. Then he selects the corresponding card. The estimators at this stage do not discuss their estimates. This step is done individually by each estimator.
4. In this step, all cards are simultaneously turned over. This way all the participants can see each others estimates.
5. If all the estimates are equal then that value is selected for cost estimate.
6. However if there is any disparity in the estimates then the participant with the highest value and those with lowest values explain their concerns and the team discusses the scope of the story again.
7. After this discussion, the steps 3and 4 are repeated until all the participants have agreed to a common estimate.

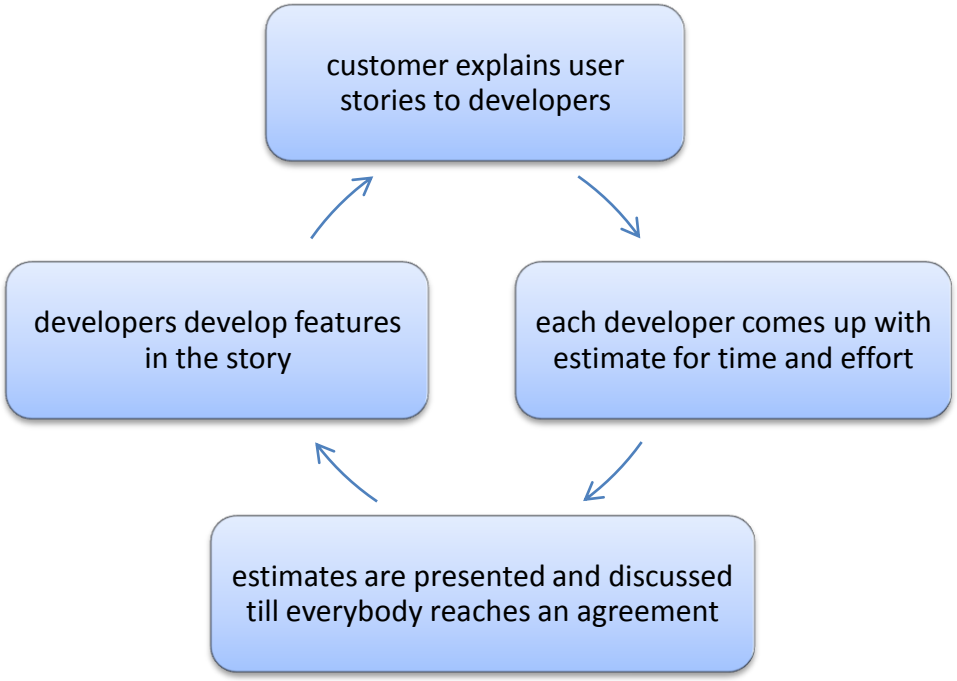

Fig 8. Planning Poker

Shortcomings of Planning Poker

Based on a research [24][25], 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:

$$MRE = \frac{|EstimatedCost - ActualCost|}{ActualCost}$$

Mean MRE is used to quantify the accuracy for the complete model. Based on the estimation data collected from the enterprise for planning poker [24], 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 planning poker.

- **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.

If the user requirement documents are very well defined then we can easily use popular effort estimation measurements in software industry. In Agile Software Development methods requirements are subject to change and there is no standard way of specifying them. User requirements in agile methods are defined as user stories and are collected in backlog.

Agile methods have typically less detailed processes based on their values and principles. Such activities are really light-weight in terms of processes for planning, sizing and effort estimation as well as project management. Most of the measuring practices in traditional software development methods are not usable in agile methods directly. They have to be customised for use in the particular company or a specific type of projects. In the next chapter we describe in detail some these modified estimation approaches under research.

Chapter 5 Recent Software Cost Estimation Approaches for Agile Development

Introduction

Due to uncertainty in requirements, the traditional estimation approaches cannot be used for estimation of size, cost or duration in agile software development projects. However as we have seen from the previous chapters that agile estimation relies mostly on analogy based methods or expert judgement based methods for cost estimation.

Following are some of the current practices of agile software development cost estimation techniques under research

1. **SWOT based estimation model given by- Ziauddin, Shahid Kamal Tipu, Shahrukh Zia, "An Effort Estimation Model for Agile Software Development",** ***Advances in Computer Science and its Applications (ACSA),*** **2012**

The authors proposed a multidimensional view to produce accurate and effective estimates using a *SWOT* according to Internal vs. External influences. They used data collected from past projects combined with mathematical formulae to develop a model to estimate the effort, project duration and cost. The model predicts Completion time and cost for agile software project. They identified the key differences between team organisation in agile and traditional development approach as follows:

1. *Agile teams are "Whole" :*

   It is an XP practise which implies that the team members have the requisite skills between themselves only. Within the members of the team they have the required testing skills, interfacing skills, UI designing skills, database skills, translation skills etc. and do not need any external teams dependency to complete the project

2. *Agile teams are formed of generalizing specialists:*

   A generalizing specialist is someone who has one or more technical specialties e.g. Java programming, project management, database administration.

3. *Agile teams are stable:*

   This means that any significant change in the team structure or organisation can affects on the project performance.

The authors found that the scrum practitioners used a comparative scale for estimating the development effort. That is the development cost is estimated comparative to previous projects; where the scale is often a Fibonacci scale [1,2,3,5,8].This means that the story ranked 3 is approximately thrice as costly to develop than the one ranked 1.

The authors argued that this method of prediction by relative estimation does not take into account the underlying elements that affect effort and uncertainty.

Effort estimation is based on user story size and its complexity. Using these two vectors, effort of a particular User Story is determined using the following simple formula:

ES= Complexity x Size

For project cost estimation the concept of agile velocity was used. *Velocity* is calculated as

$$Velocity\ Vi = \frac{units\ of\ work\ completed}{sprint\ time}$$

This is the observed velocity or initial velocity Vi.

The authors proposed a mechanism to optimise the velocity value by two factors:

i. The Friction or consistent forces that are a constant drag on productivity and reduce Project Velocity.
ii. The Variable or Dynamic Forces that decelerate the project or team members and cause the Project Velocity to be irregular.

Friction Factors are the external factors such as environment factors, process factor, team dynamics etc which negatively impact the productivity hence increase the cost. The aim is to reduce or minimise the friction factors.

The friction value FR is given as a product of all individual friction factors values.

$$FR = \prod FF$$

Variable Factors are internal factors which are unpredictable and unexpected. They are for a brief time and introduce a little irregularity in the velocity hence affect the cost. These forces need to be made consistent and predictable as much as possible so that their effect on the agile velocity can be predicted at the time of estimation.

Factors such as re-organisation of team, change in management, new tools or process, unclear requirements, personal issues of development team etc can be considered under variable factors.

The dynamic force DF is given as a product of all individual variable factor values.

$$DF = \prod VF$$

Now the authors gave the following formulae for optimised velocity

$$D = FR \times DF$$

$$V = Vi^{D}$$

Where V= optimized velocity, D = deceleration.

The following formula is given for estimating development cost:

$$COST = NetRatio \times \frac{ES}{V}$$

Here ES refers to story point values.

Using empirical data the authors calibrated their model and found the net ratio to be 1.68.

The authors have also measured the estimation accuracy by performing experimental analysis on data of previously developed projects from different software houses. They found that MMRE for estimation of cost comes out be 61.90%.

This model also provides for uncertainty in the measurement by introducing a *"Span of Uncertainty"*.

The estimator can never be 100% sure of their estimates, hence they have a *confidence level CL*. It indicates how much confidence they have in their estimates, how sure they are of their understanding of the magnitude of project factors. CL is input in terms of % value. Typically it ranges between 80% to 95%. Using the CL confidence level indicator, the model helps find the variation range in the predicted cost. The lower bound of this range is *Optimistic Point* and the upper bound is *Pessimistic Point*.

2. **Chandrasekaran, R. Lavanya S., and V. Kanchana. "Multi-criteria approach for agile software cost estimation model",** ***Proceedings of APA*****, 2007**

The authors have modelled a software cost estimation process with a number of constraints imposed by stakeholders and environmental characteristics, thereby satisfying multitudinous criteria using concurrent constraint programming. The proposed work is to focus on the various factors affecting the people-oriented environment. The authors have argues taht in agile environment, the development cost of a software project is dependent to a great extend

on the people and management issues. The cost factor for agile software is based on a multiple-criteria approach.

The conceptual model describes the idea of arriving at the criteria set required to emulate the agile environment. The major quality-attributes relating to the each of the agile manifestoes and affecting the agile software are identified. Numerical weights are attached to them that represent the effect of the attribute on the product quality and time of completion. the authors have called them *Quality Weights (QW)* and *Time Weights (TW)* respectively.

These attributes are the cost drivers and by combining the agile manifestoes with the various quality and time attributes, a particular set of the attribute levels are derived that forms the criteria for estimation.

The below tables show the attributes and their time and quality weights as given by the authors.

Table 3.High priority attributes and their weights as given by the model.

High Priority Attributes	**Quality weight *QW***	**Time Weight *TW***
Communication skills	5	-4
Proximity of team	3	-4
Feedback	1	5
Courage	1	-3
Managerial skills	5	-5
Consistent working	2	-2
Technical ability	3	-2
Debugging capability	2	-2
Reliability	3	-2
Function points	3	4
Ease of use	4	3
Early deliver	5	-2

Table 4.Low priority attributes and their weights as given by the model.

High Priority Attributes	**Quality weight *QW***	**Time Weight *TW***
Process maturity	4	-3
Toolavailabilty	4	-2

Tool familiarity	3	5
Conservativeness	1	3
Training	4	4
Planning	4	-3
Pages of documentation	2	3
Project complexity	1	4
Other expenditure	2	3
Documentation resources	2	3
Documentation period	4	3

The value of *Quality Factor* QF is given by

$$QF = \sum (QWi \times Li)$$

Where QWi and Li refer to the quality weightage and level of each of the attributes.

The value of *Time Factor* QF is given by

$$TF = \sum (TWi \times Li)$$

Where TWi and Li refer to the time weightage and level of each of the attributes.
Once these are determined, the model proposes the estimated Cost as follows:

$$CF = TF \times QF$$

Where CF = cost factor,

TF = time factor

QF = quality factor

The authors found that their model was well suited to small and medium sized development teams which use agile development methodology. But the major drawback of this is that the accuracy of the estimation depends entirely on the mathematical representation of agile manifestoes.

3. **Asnawi, Ani Liza, Andrew M. Gravell, and Gary B. Wills. "Factor analysis: Investigating important aspects for agile adoption"** ***AGILE India (AGILE INDIA)*****, 2012. IEEE**

The authors used the Factor Analysis technique to identify/propose 15 factors. They conducted a survey, and evaluated the practices having significant contributions in several IT areas such as process/governance, quality assurance, iterative and incremental development and team communication but not directed at project performance aspects such as cost, quality, deadlines and scope.

Factor analysis is conducted to identify the clusters of the variables (or items) and how they are inter-related to produce factors. The clusters of variables resulting from this analysis can serve as a reference to the practitioners planning to adopt the methodology.

To see how Agile adoption variables in our study are inter-related, questions regarding Agile adoption were asked to software practitioners. The suitability and appropriateness to conduct factor analysis with the data need to be checked. One of the statistical measures used to identify this is called Kaiser-Meyer-Olkin (KMO). It is a measure of sampling adequacy which ranges from 0 to 1.

Values between 0.5 and 0.7 are mediocre, values between 0.7 and 0.8 are good, values between 0.8 and 0.9 are great and lastly values above 0.9 are superb. A KMO with 0.6 is suggested as the minimum value for a good factor analysis. If the value yields more than 0.7, then the correlation on the whole are sufficient to make factor analysis suitable.

The authors on measurement found a KMO value of 0.755 was obtained from their data.

From an initial 27 factors they have extracted 8 factors. The Eigen values ranged from 0.093 to 7.852. The eight extracted factors and their related variables are described as follows:

Table 5. Variables and their loadings

VARIABLE	**LOADING**
FACTOR-1 Level of Involvement of the developer and impact of their opinion	
Responsibility of the developers towards organisation's Agile mission	0.816
Involvement of developers in setting goals for Agile activities	0.805

Importance of identifying project scope and suitability of agile in the project	0.674
Transparency and encouragement to developers	0.497
Allowing of interpersonal interactions among the developers	0.564
FACTOR-2 Organisational Culture and People Related Aspects	
Presence of people of different ethnic and racial backgrounds	0.845
Use of English as communication language	0.810
change of mindset when using Agile practices	0.434
FACTOR-3 Customer Involvement when Practicing Agile methods	
Involvement of customers in setting goals for Agile activities	0.680
Requirement of special skills for agile practises	0.656
Resposibility of the customers towards fulfilling the organisation's Agile goals	0.615
Customers knowledge of Agile Practises	0.556
FACTOR-4 Benefits/Impact of using Agile methods	
Focus on customers' satisfaction when using Agile methods	0.881
Efficiency in the software development process achieved by collaboration between both parties i.e. customers and developers.	0.867
Agile Developer's Morale	0.585
Quickness of delivery of results by Agile Methods	0.495
FACTOR-5 Importance of training and learning	
Training for Agile methods	-0.879
continuous learning helping in knowledge transfer when using Agile methods	-0.811
FACTOR-6 Importance of Technical and Technological Aspects when using Agile	
Suitability of Agile methods for that specific project and technology	-0.943
Importance of tools for supporting the usage of Agile method	-0.507
emphasises on achievement and goal accomplishment	-0.414
FACTOR-7 Importance of Sharing, Knowledge. etc	
Personal interaction between team-mates	0.614
Limitations regarding Knowledge about Agile Practices	-0.530
FACTOR-8 Team Commitment and Clarity of Purpose	
Clarity in division of knowing roles and responsibilities of each member	0.694
attitude such as team spirit and team commitment required from everyone	0.515
Early Delivery Time requirement	0.493

The authors found organisational and software developers' involvement as the top factor when using Agile methods. This result also shows that language is one of the important aspects when adopting Agile methods. In terms of the impact that Agile can deliver, high loadings (greater than 0.8) were found to be in customer satisfaction and the ease of software development as a result of collaboration between developers and customers.

These extracted factors explain the practitioners' negative perceptions about the importance of agile development technical aspects in adoption process, valuing more human aspects such as customer satisfaction and collaboration among developers and customers.

The top factors identified in the study are shown in terms of (i) developer involvement and organisation-related aspects, (ii) cultural and people related aspects and (iii) customer collaboration and the need for professional skills when using Agile methods. In addition, factor analysis discovered that practitioners disagreed about the importance of the technical aspects of Agile. Moreover, the study also can help other adopters from to understand and see the suitability of Agile methods for their organisations.

4. **Santos MA, Bermejo PHS, Oliveira MS, Tonelli AO "Agile Practices: An Assessment of Perception of Value of Professionals on the Quality Criteria in Performance of Projects",** ***Journal of Software Engineering and Applications*****, v. 04, p. 700-709, 2011**

The authors have analyzed practitioners' perceptions of the impact of agile practices, using factor analysis and considering different methodologies such as Feature-driven development (FDD), Extreme programming (XP), Scrum, Crystal Methodologies, Dynamic systems development method (DSDM), Test-driven development (TDD).

They investigated about which are the main agile practices adopted to ensure the benefits in quality of software projects using agile methodologies.

Overall, the available studies focus on analyzing factors and attributes is related to agile principles and not on the applicability of practices. The authors analysed from the perceived value of various stakeholders in the process of software development, the relationship between the use of agile practices and quality of software products.

Another important point raised is the limitation of these studies to evaluate projects using specific methodologies and more popular, such as XP and Scrum. The magnitude and diversity of practices and agile methodologies it's disregarded in most of the studies.

Therefore, the investigation of the perceived value of the users those agile practices in their software development environment is presented as an appropriate objective since the article proposes to seek understanding of the impact and positive effect of the adoption of agile practices in software projects as a way to get quality on the final software product.

The authors collected a sample of 109 complete evaluations and found what aspects should be considered for achieving higher quality in software products. In general the practices to be used in order to reduce cost and flexible scope while using agile development methodologies can be given by improving of the following four development aspects: (a) team abilities, (b) requirements management, (c) quality of the code developed, and (d) delivering of software within budget and in-time

The authors gave a set of six latent root criteria which are given below and depicted in fig 9.

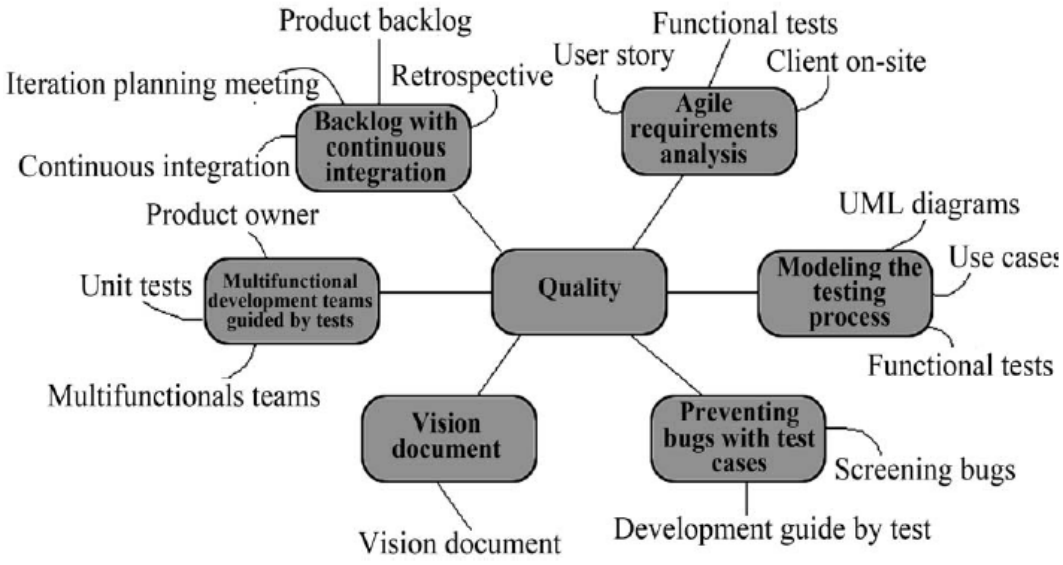

Fig 9. Agile grouped into criteria that represent user's perception of value and quality

i. Backlog with continuous integration:

It represents practices related to the phase of planning of the features of the project, where the team sets during the planning meeting of the iterative cycle, the implementations of higher priority and which deliver customer value. The practices had higher factorial weights This justify a positive perception of value from the respondents regarding the creation of a prioritized list of features in the iteration planning meeting, that must be worked and integrated in small releases

and continuously evaluated qualitatively at the meeting of the Iteration Retrospective, in order to create a improve plan for the next features to be worked.

ii. Agile requirements analysis:

This refers to the perception of practices related to the execution and analysis of test of features specified in the users stories. A User story is an agile practice where the requirements are specified from the customer's point of view, in a simple language and description. In this case, the data indicate which this practice plays an important and active role in the definitions of the projects and that the features created from the stories users are properly tested through the use of functional tests.

iii. Modelling the testing process

This has high significance in practices related to the modelling of testing phase. The practices had higher weight factor justifying the positive perceived value of the respondents in the generation of use cases and UML diagrams to build test cases adapted to the agile process in their companies.

iv. Preventing bugs with test cases

Use of test cases for the correction of errors arising from the acceptance tests. The practices that justify the development of test cases to implement the features ill-defined in the planning phase and not properly functional identified in screening of errors of acceptance tests by the user.

v. Vision document

This represents the use of a vision document as an artifact which reports system technical's perspectives, process preceding the analysis of the domain model. In this case, the Vision Document, despite being built using simple language, yet has a more technical aspect of a user story.

vi. Multifunctional development teams guided by tests

This sixth factor represents factorial weights in practices related to the formation of a cross-functional team which implements unit tests of the more valuable features. The values of features are defined in accordance with the customers or

the Product Owner. A team with different abilities to execute tests could bring positive results because they will search for failures in the sys- tem from different aspects. The practices had higher factorial weights justifying the positive perception of value of the respondents to practice cross-functional teams including the customer in a more representative way.

The authors showed that agile practices combined into factors used in the various phases of the project, can contribute positively in achieving quality in three aspects:

a) developed code in terms of maintainability, reusability and re-factoring of code.
b) Higher involvement of stakeholders as they present the business point of view for the project.
c) Management of the uncertain and changing requirements

They also suggested combinations of widely used agile practices in the software market. These can be designed for suitability and compliance of quality standards in software projects can lead to a performance positive.

5. **J.-M. Desharnais and L. Buglione, "Using the COSMIC method to estimate Agile user stories,"*12th International Conference on Product Focused Software Development and Process Improvement, Torre Canne, Brindisi, Italy,* 2011.**
   COSMIC, "Guideline for the use of COSMIC FSM to manage Agile projects" 2011.

They proposed a procedure for using COSMIC Function Points in agile methods and assessed it in a real project. It involves some degree of guess estimation on some of the user stories; however by eliciting requirements from the user stories and focusing on high quality of documentation, this methods can be helpful.

The authors proposed an approach based on COCOMO using cosmic measurement method at user story level in addition to quality of documentation. They also demonstrated that their approach can help planners know better why the global effort changes along with time.

The proposed approach can be given by the following flow diagram

Each of the steps is described below.

The COSMIC method is used to provide a standardized method of measuring the functional size of software. Firstly user requirements are taken in the form of User Story (US). A user

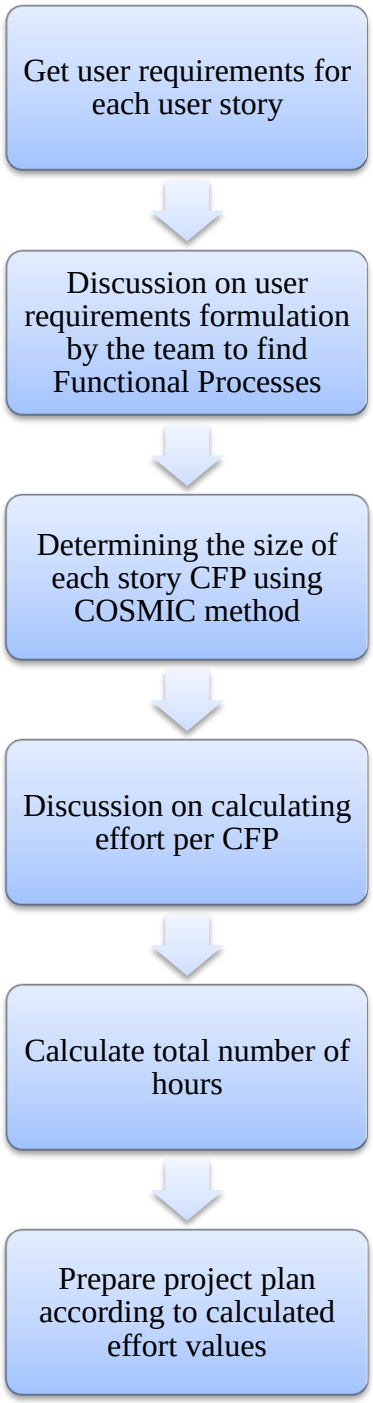

Fig 10. Procedure to obtain effort in hours for a story

story describes a feature of the software. Each story is formulated in one or two sentences in the language of the customer. This is only a short broad level description of the user story.

Then these user stories are formulated in terms of technical implementation and their functional process are determined. This is the mapping phase of the cosmic method. The mapping procedure is given in table 5 below.

Table 6. Procedure Steps to apply COSMIC method – Pt. 1

MANAGEMENT STRATEGY (MS)	MS.01 Defining Purpose of measurement
	MS.02 Defining Scope of measurement
	MS.03 Identifying Functional Users
	MS.04 Identifying level of granularity
MAPPING (MA)	MA.01 Applying generic software model
	MA.02 Identifying functional processes
	MA.03 Identifying objects of interest and data groups

COSMIC measurement method raises some questions about the project information such as:

- how many functional processes within a story
- is the story imply a change in the database
- what is the trigger of the functional process

As the project will go on, and more information is coming in, there is a possibility to have a number of changes in the software product.

Next step is the determination of the functional size of each US n terms of CFPs

Measurement of the stories is done by using the COSMIC method, which gives the size of each functional process in terms of COSMIC Function Points or CFP from the information available at that time. This completes the Mapping phase as shown in table 5. Remaining steps to apply COSMIC method are shown in the following table:

Table 6. Procedure Steps to apply COSMIC method – Pt. 2

MEASUREMENT (ME)	ME.01 Identifying Data Attributes
	ME.02 Identifying Data Movements
	ME.03 Application of Measurement function
	ME.04 Aggregation of Measurement Results

After the functional size for each US is determined, effort per CFP value needs to be determined. Experienced members of the development team discuss and find an agreement on the value of effort per functional point. this value can also be derived from a repository of previously implemented projects, selecting their more feasible PDR(productivity) in terms of effort/CFP values after applying some filters by the needed project characteristics e.g. development type, programming language(s), application domain, etc.

Next step is to apply those PDRs to each US and therefore summing them within the boundary of the overall project. PDR values will be multiplied by the number of measured CFP for each US, and such value will return the needed effort for that US. Then in order to calculate the total effort of the project in terms of person-hours (p/hrs), the sum of such calculated values will be taken.

$$Overall\ Project\ Effort(in\ person-hours) = \sum_{i=1}^{n} CFP_{USi} \times E_{USi}$$
$$= [(CFP_{US1} \times E_{US1}) + (CFP_{US2} \times E_{US2}) + \cdots + (CFP_{USn} \times E_{USn})]$$

Where *n* is the number of user story,

CFP_{USi} is the COSMIC Function Point for i^{th} user story

E_{USi} is the effort per CFP for i^{th} user story

The authors also demonstrated using an example that as a result of such measurement, needed effort for the project will be calculated more accurately. So, effort plans, budget plans and resource plans of the projects will be more reliable with this approach. Additionally, there is still a little amount of guess estimation in determination of the effort per CFP for some US. In fact, this guess estimate does not represent the effort level for all the US in a project, but just for one or more of them.

Based on the Desharnais' work, COSMIC published officially an agile version of COSMIC FFP for using in agile software development. COSMIC introduced first official guideline in ASD methods for software sizing based on function point. In agile COSMIC, according to the agile features, some modifications were made. Each User Story is defined as a single FP. CFP does not require detailed specifications and also, mapping between CFP and USP is easy and understandable. CFP method measure functional user requirement and estimation is derived in terms understood by users of the software. In CFP four base functional components types (read, write, entry and exit) are extracted and estimation relies on these items. To satisfy requirement changes requested by customer, functional size of each User

Story could include changes to a previously released User Story. The estimation is basically done in the beginning of iterations and it works on the updates over previous predictions. Using this method, any change in any of the user requirements could be calculated which will

happen in the next iteration. Hence, based on adding, changing or cancellation of any data movement, change in the size of software is measured and the cost increments are deduced from that. Here the first estimate is of prime importance as any error in the first estimate will propagate and increase each of the subsequent iterations.

6. **P. Abrahamsson, I. Fronza, R. Moser, J. Vlasenko, and W. Pedrycz, "Predicting development effort from user stories",*International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE*, 2011, pp. 400-403.**

The authors proposed a model for predicting development effort based on user stories. Their method relies on predictors which are extracted from completed user stories and which will be used for next stories. The approach is well suited for agile software projects where requirements are developed along with the product. Initial requirements are sketched in a rough manner only.

Given a set of user stories the users define a set of predictors that can be extracted automatically from a user story. No particular structure or format is required for the content of the story. Next the models are built to predict then implementation effort of a user story.

The results are then compared to estimation based on design metrics.

The model predicts the effort for each iteration defined by asset of user stories, hence it is naturally fitting for Agile development practices. Also the data required for the estimation process is user stories only which are readily available after the planning phase and before development phase. Since it is not dependent on prior or expert knowledge, novice users and new teams can also use it for estimation process.

A schematic view is given in Figure 11.

Initially a set of completed user stories are available. Predictors are extracted from the user stories.

Predictors extracted from user stories are:

i. Number of characters – a higher value depicts a more complex user story because developers need more text to define complex functionality. Hence a higher number of characters mean more development effort.

ii. Presence of keywords – the authors have given a list of 15 keywords. The user stories are checked for presence and frequency of these keywords and this is stored as binary values.

iii. Priority – the team discussions occur with customers to decide on priority order of user stories; 1 indicating the highest priority and 4 denoting leastd priority. The authors considered high priority user stories to be requiring more effortthan low priority stories.

The 15 keywords identified by the authors are –

Gestation (Management), File, Test, Creation, Report, Tool, Possibility, Data, Modification, Visualization, Configuration, Build, To Visualize, Time, Channels.

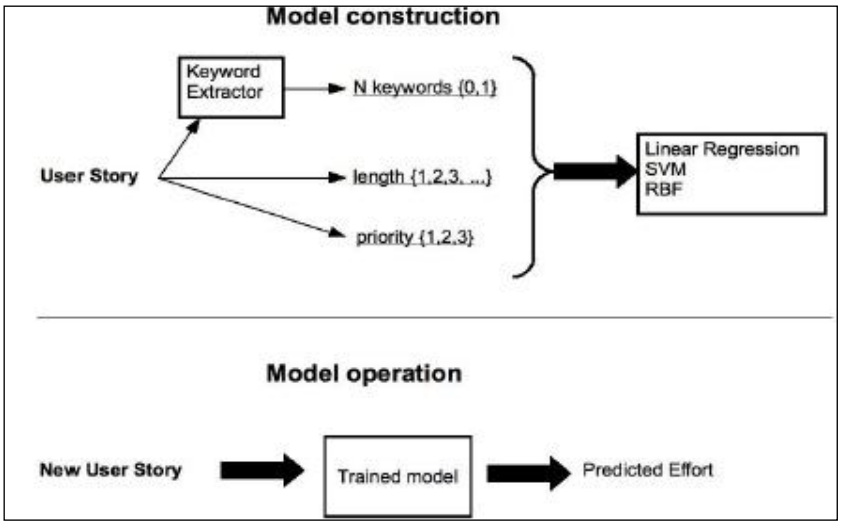

Fig 11. Effort prediction model

These predictors are used to train the model. the authors consider only algorithmic models in their estimation process. The models which can be quantitatively derived from the data such as: regression models, neural networks, Support Vector machines(SVM).

Then the model is used for estimation for new stories. Cross validation methods like leave one out (LOOV) procedure.

Such approach is well suited for Agile software projects where requirements are developed along with the project and only sketched in a rough manner. In the proposed model the effectiveness of the model is different from case to case and is based on quality and style of user stories.

By applying their proposed method to two industrial Agile software projects of very different size and structure the authors show that such effort estimation works reasonably well if user stories are written in a structured way.

Chapter 6 Proposed Cost Estimation Process for Agile Software Development Projects

In this chapter we present a cost estimation model for agile software projects which estimates the development cost by extracting key factors using Principal Components Analysis and using constraint programming to satisfy the constraints imposed by the agile manifesto. The model is then evaluated and compared with recent researches.

6.1 Overview of the Cost Estimation Model

Using inspiration from recent researches involving SWOT analysis based model and use of PCA-based models for estimation in traditional development methods, we propose a Cost estimation model as depicted in the figure12**.** The first step is to identify the factors affecting development costs by analysis of the sample data by using the Exploratory Factor Analysis with factor extraction using Principal Component Analysis (PCA). This results in generation of the coefficient matrix for each of the identified principle components. The second step is to use constraint programming for satisfying the criteria imposed by agile development environment through the agile manifesto. The Development cost is determined by using the factors and coefficients generated by factor analysis while satisfying the agile manifesto conditions by the Constraint programming. This estimation process is expected to enhance the level of visibility of cost estimation in the planning stages.

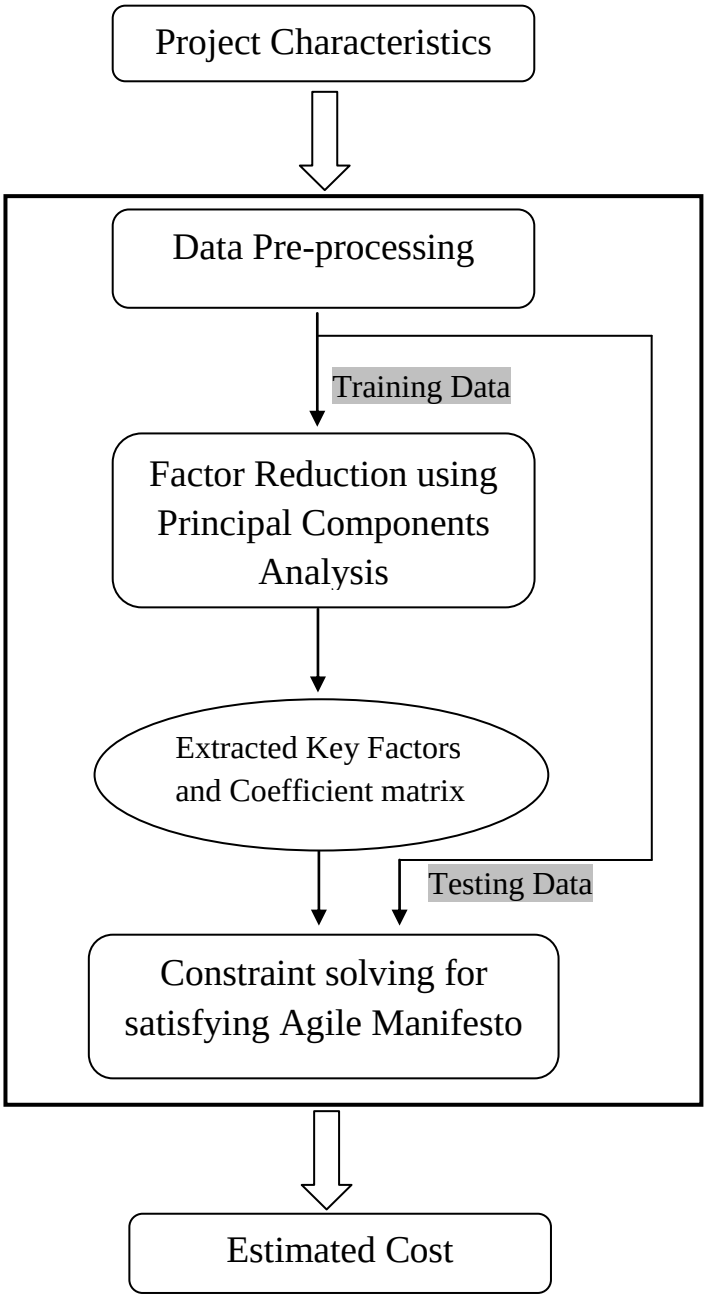

Fig 12. Process model for cost estimation

6.2 Data pre-processing

Agile paradigm considers the people factor to be more critical than the process factor. For processes to be predictable, components in it need to behave in a predictable way. However people are not predictable components, hence there exists a difficulty in predicting and quantifying software for cost estimation. Also, uncertainty, risk factors, emerging requirements and complexity issues are presented in agile as in any other traditional software development process.

Hence the qualitative data in the data set was converted into a set of quantitative values through summarizing on an N-point scale (where N is a prime number usually 3 or 5 or 7).
The next step is smoothening of data which involves treatment for missing data values. The missing data which was mostly empty was ignored; in case the missing data was less it was filled with the mean value. Next Step is removal of statistical noise and deletion of exceptional/extreme points in data.
The complete data set is divided into two parts - training data set and testing data set.

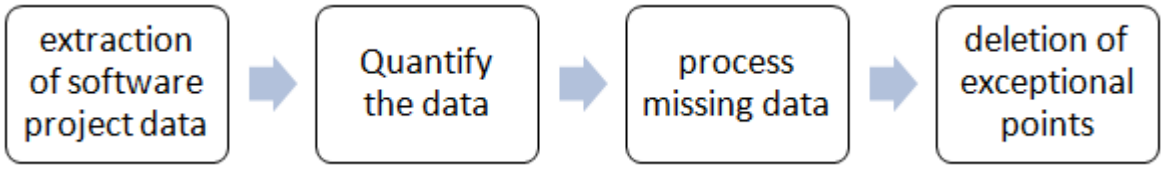

Fig13. Steps in data pre-processing

6.3 Exploratory Factor Analysis using Principle Component Analysis

Following the pre-processing step the agile development data will be analyzed using the multi-variate statistical technique Exploratory Factor analysis
In theory, EFA is a technique for exploratory data analysis consisting in data reduction or a structure simplification, to describe, if possible, the ratio of the covariance among the many variables in random and unobserved quantities named factors. Determine nature and number of latent variables that account for observed variation and co-variation among set of observed variables. The selection of this technique is motivated by the argument that variables can be grouped based on the correlation degree, in other words, the strength and direction of the relationship among variables. The agile development data contains data from multiple projects. This data contains values of recorded attributes for each of the software project. The recorded attributes are:

Table 8. Attributes for cost estimation in data-set

Software size IDE Productivity Development Platform	Application Type Programming Language Operating System Team Size

Language Type	CMMI
Data Base System	Architecture
Organization Type	ISO
Hardware	Type of Server
Following process model	Package Customization
Training	Project complexity
Technical ability	Function points
Planning	Reliability
Debugging capability	Pages of documents
Client/Server	Risk taking
Communication skills	Managerial skills
Process maturity	Ease of use
Proximity of team	Documentation resources
Tool availability	Early delivery
Feedback	Documentation period
Tool familiarity	Other Expenditure

The objective of the use of factor analysis is to explore the sample data to generate future hypotheses about development costs involved in software development using agile practices. Factor extraction is done using statistical analysis method of Principal Component Analysis (PCA). This method is selected because it explains the total data variance represented in the variables in data reduction to factors. It is also the model most used in Factor Analysis, especially in statistical packages such as Statistical Package for the Social Sciences (SPSS) and Mathematica.

6.3.1 Principal Component Analysis- PCA

PCA is the statistical technique which is used to reduce the dimensionality of a data set which has a large number of inter-related variables, while also retaining as much as possible of the variation present in the data set. This is achieved by transformation of the given data into a

new set of variables, the PCs which are uncorrelated, and which are ordered so that the first few retain more variation than the rest of the components.

This definition means that the PCA transforms the data to a new coordinate system ensuring the highest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on in descending order.

Using the PCA method will reduce multiple related factors into a few comprehensive and linearly un-correlated variables which can include the most information of the agile development data.

In particular, the EFA model is represented by following equations :

$X_1 = \ell_{11}F_1+\ell_{12}F_2+ \ldots+ \ell_{1m}F_m+\varepsilon_1$

$X_2 = \ell_{21}F_1+\ell_{22}F_2+ \ldots+ \ell_{2m}F_m+\varepsilon_2$

:

$Xp = \ell_{p1}F_1+\ell_{p2}F_2+ \ldots+ \ell_{pm}F_m+\varepsilon_p$

the coefficient ℓij is named the (factor) loading of the *i*th variable in the *j*th factor, where the letters *i* and *j* are integer index 1, 2, 3 ... , and $L_{(p \times m)}$ is the factor matrix with loadings. In this context, the factor analysis model assumes that these variables show a linear relationship with the new variables F_n, where n = 1, 2, 3 ... m. The vector $\varepsilon_{(px1)}$ represents the random errors associated with measurements.

In statistical analysis works, the variables that have factor loadings of 0.55 or higher are usually considered.

The principle components are extracted from the sample data by the following steps:

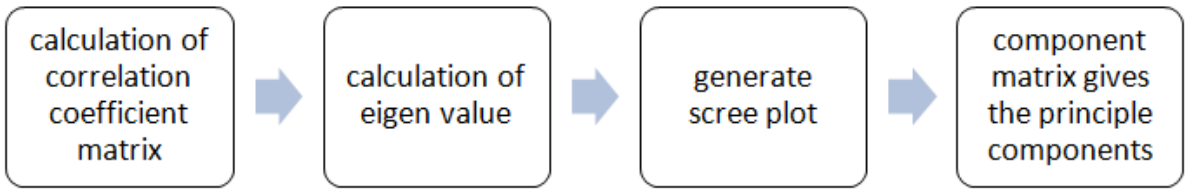

Fig 14. Steps in Exploratory Factor analysis

6.3.2 PCA based Factor Analysis

From the processed training data set we obtain the component correlation matrix. The Component Correlation matrix contains the Factor loadings for each component which

explains the co-relation between the corresponding component and the cost of development. These are essentially the eigen values of the covariance matrix of X for each component. Communalities are the variance in observed variables (here it refers to project development cost) accounted for by common factors. Using these we draw a scree plot.

A scree plot is a graphical display of the variance of each component in the dataset which is used to determine how many components should be retained in order to explain a high percentage of the variation in the data. The plot shows components at the X axis and the corresponding Eigen values at the Y-axis. The variance for the first component and then for the subsequent components, it shows the additional variance that each component is adding.

The components whose values are below a predefined threshold value are dropped and those above the threshold value are considered as principal components.

The Principal Components are identified through accumulation contribution, communalities and scree plot as is depicted in the following flowchart.

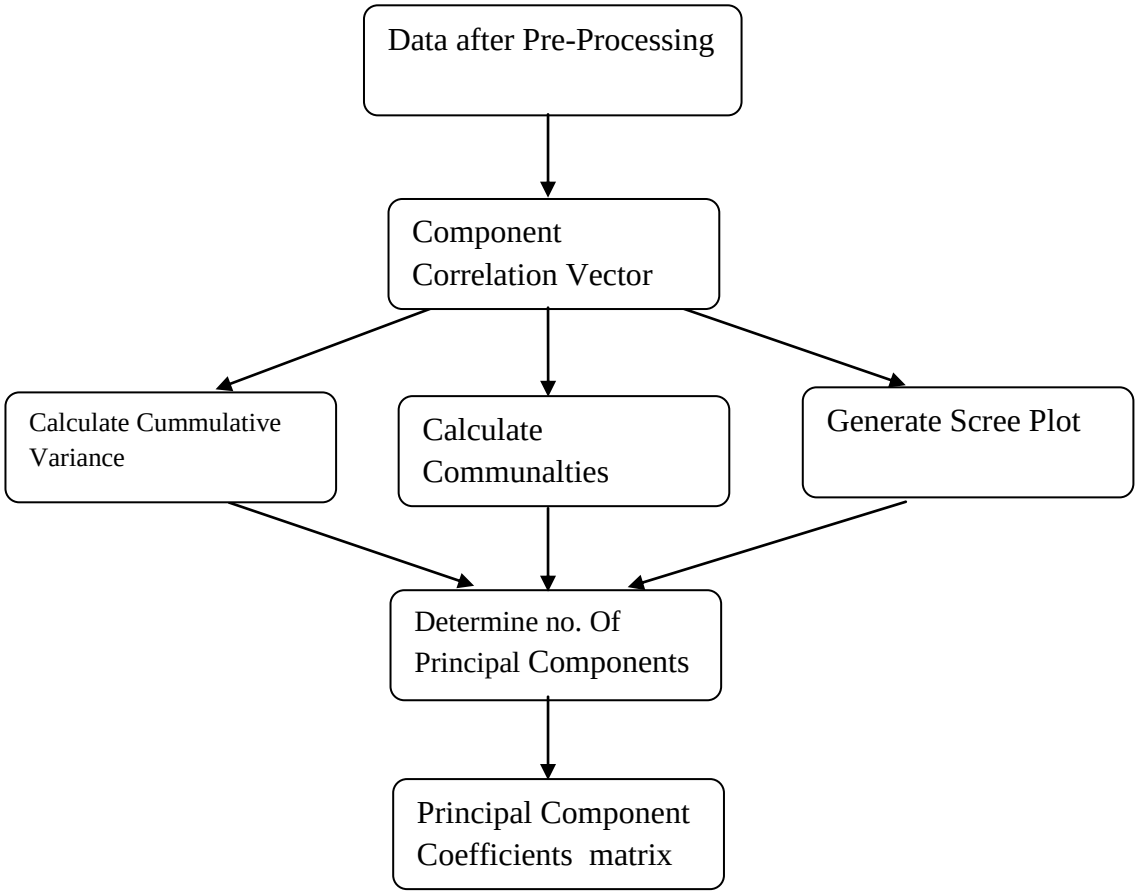

Fig. 15 Steps of Factor Extraction using Principal Component Analysis

6.4 Constraint Programming for satisfying Agile Manifesto Criteria

On obtaining the set of principal components/attributes and the score coefficients matrix we have to determine the criteria set required to emulate the agile development environment.

The agile manifestoes impose certain constraints on the factors that are to be concurrently solved to obtain these criteria. These are:

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

In the software development environment different possible combinations of the components' coefficient values are equally likely to exist. The objective of this step is to formulate a set of criteria for an agile environment from this huge domain. Different issues affecting the people-oriented and adaptive agile environment are considered to come out with a set of criteria based on which the estimation process is carried out. Constraint Solving follows the following steps:

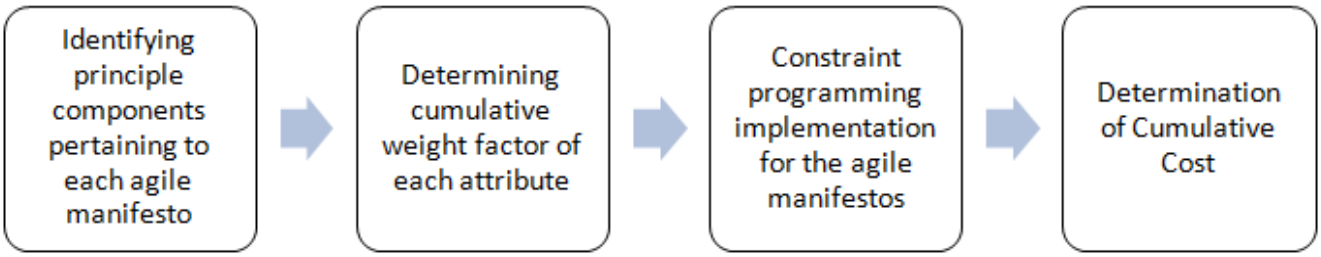

Fig16. Steps in Constraint solving

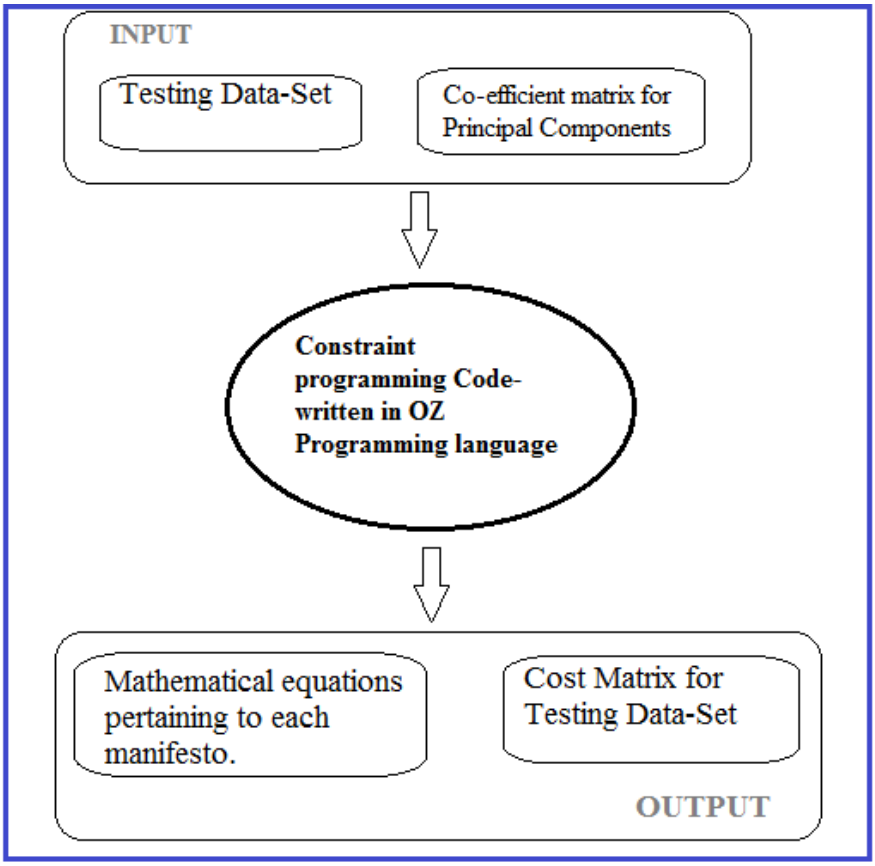

Fig.17 Process model for constraint programming

In order that the estimation best fits the agile environment, the four manifestoes of the agile are given prime importance. The PCA outputs key factors extracted and their corresponding coefficient values. The extracted factors are classified under the agile manifestos as applicable. Each extracted factor corresponds to one or more manifesto.

Let there be N extracted factors denoted as $f_1, f_2, f_3 ... f_N$ *with corresponding coefficient values as* $C_1, C_2, C_3 ... C_N$

Now we classify these extracted factors according to each of the agile manifesto and calculate the cumulative factor value for each manifesto.

Let factors f_1, f_4, f_6, f_9 be applicable to the first manifesto. These are then denoted as $f_{11}, f_{12}, f_{13}, f_{14;}$ and their respective coefficient values as $C_{11}, C_{12}, C_{13}, C_{14}$.

Similarly f_{21}, f_{22}, f_{23}.. denote the factors which are applicable to the second manifesto and C_{11}, C_{12}, C_{13} are their corresponding factor coefficients;

f_{31},f_{32},f_{33}.. denote the factors which are applicable to the third manifesto and C_{21}, C_{22}, C_{23} are their corresponding factor coefficients;

and f_{41},f_{42},f_{43}.. denote the factors which are applicable to the fourth manifesto and C_{41}, C_{42}, C_{43} are their corresponding factor coefficients.

Next we calculate the cumulative factor value for each manifesto as a summation of individual products of the value of the corresponding attribute and the coefficient value.

Cumulative Factor Value for manifesto-1 F_1 is given as:

$$F_1 = (f_{11} \times C_{11}) + (f_{12} \times C_{12}) + (f_{13} \times C_{13}) + (f_{14} \times C_{14})$$

It can be also expressed as :

$$F_1 = \sum_{i=1}^{N_1} (f_{1i} \times C_{1i})$$

Where N_1 denotes the number of factors which correspond to manifesto-1.

Similarly Cumulative Factor Value for manifesto-2 F_2 is expressed as:

$$F_2 = \sum_{i=1}^{N_2} (f_{2i} \times C_{2i})$$

Where N_2 denotes the number of factors which correspond to manifesto-2.

Cumulative Factor Value for manifesto-3 F_3 is expressed as:

$$F_3 = \sum_{i=1}^{N_3} (f_{3i} \times C_{3i})$$

Where N_3 denotes the number of factors which correspond to manifesto-3

Cumulative Factor Value for manifesto-4 F_4 is expressed as:

$$F_4 = \sum_{i=1}^{N_4} (f_{4i} \times C_{4i})$$

Where N_4 denotes the number of factors which correspond to manifesto-4.

The development cost is estimated as a product of the cumulative factor value of each of the manifesto as –

$$\boldsymbol{COST = F_1 \times F_2 \times F_3 \times F_4}$$

Constraint solving is done using the OZ programming language. OZ is a multi paradigm language that is designed for advanced, concurrent, networked, soft real-time and reactive applications that is ideal to solve the constraint base estimation problem. The OZ constraint programming allows to pre-define the set of constrains that must be satisfied constantly rather than having to write the methods to maintain the relations

Chapter 7 Implementation and Results Analysis

7.1 Introduction

In this chapter we will implement the proposed estimation model on the sample data set and demonstrate the steps of the cost estimation process. The model is evaluated by finding out the mean magnitude of relative error observed (MMRE) and compared with two of the existing models- SWOT based estimation model and multi-criteria based approach.

7.2 Data-Set Description

The proposed two step model for cost estimation in agile software projects was used on the data-set which is used in research in the domain of agile development methodologies and agile studies at the 'Centre For Systems And Software Engineering' at the 'School of Engg, University of South California'. The dataset contains knowledge about software projects that are *'standardised, verified, recent and representative of current technologies'*.

It is a repository of the software development project data from about 250 development projects using agile technologies. In other words ait is a record of values of 40 attributes for 250 projects. The data is collected from reputed software development firms in various countries- Switzerland, USA Australia, Netherlands, Spain, China, Finland, France, Germany, Italy, and Japan.

The data is collected primarily from middle-level to big level teams with a team size ranging from 25 to 70 persons. The projects considered are also of varying size, complexity and costs. The suitability and appropriateness of the data to conduct factor analysis should to be checked. This validation of data used for analysis has been tested by the Kaiser-Meyer-Olkin (KMO) method.

The KMO method is used to measure sampling adequacy and it ranges from 0 to 1. Values between 0.5 and 0.7 are mediocre, values between 0.7 and 0.8 are good, values between 0.8 and 0.9 are great and lastly values above 0.9 are superb. A KMO with 0.6 is suggested as the

minimum value for a good factor analysis. If the value yields more than 0.7, then the correlation on the whole are sufficient to make factor analysis suitable [35].

The KMO value founded was 0.816, which according to research, corresponds to a data-set which is of good quality and suitable for analysis.

A few examples of the records in the data set are-

Table 9. Some project data from the data set used

Project Attributes	**Project 1**	**Project 2**	**Project 3**	**Project 4**	**Project 5**
Software size	113	293	132	60	16
IDE	3	3	2	0	6
Productivity	0.88	0.88	1	0.75	0.88
Development Platform	4	4	3	1	1
Language Type	2	2	0	0	1
Data Base System	7.2	7.2	5.4	1.8	1.8
Organization Type	6	6	4.5	1.5	1.5
Hardware	1.1	1	0.91	1	1
Following process model	1.24	1.1	0.91	1.24	1.24
Training	1.07	1.07	0.94	1	1
Technical ability	1.19	1	0.86	1.19	1
Planning	1.1	0.9	0.9	1	0.9
Debugging capability	1.15	0.97	0.83	1.15	0.97
Client/Server	5.5	5.5	4.2	1.4	1.4
Communication skills	4	4	2.5	-0.5	-0.5
Process maturity	8.68	7.7	6.37	8.68	8.68
Proximity of team	44	44	27.5	-5.5	-5.5
Tool availability	0.72	0.72	0.63	0.67	0.67
Feedback	3.6	3.6	2.7	0.9	0.9
Tool familiarity	2.15	2.15	1.89	2.01	2.01
Application Type	9	7	2	8	3
Programming Language	4	4	3	1	1
Operating System	14.4	14.4	10.8	3.6	3.6

Team Size	66	66	50	17	17
CMMI	2	2	5.5	2.5	2.5
Architecture	3.5	1.5	2.5	2.5	3.5
ISO	3	3	3.5	3.5	3.5
Type of Server	3	3	2	1	1
Package Customization	1.17	2.17	3.17	4.17	5.17
Project complexity	0.95	0.74	0.21	0.84	0.32
Function points	1130	2930	1320	600	160
Reliability	0.88	1.88	2.88	3.88	4.88
Pages of documents	4500	3500	1000	4000	1500
Risk taking	-0.001	0.009	0.019	0.029	0.039
Managerial skills	30	30	23	8	8
Ease of use	0.37	0.37	0.20	0.28	0.28
Documentation resources	108	84	24	96	36
Early delivery	1	1	1	1	1
Documentation period	6.4	4.9	1.4	5.6	2.1
Other Expenditure	14.83	11.63	1.77	1.74	0.24
Actual Cost	2040	1600	243	240	33

The complete data-set is provided in Annexure-2

This data can be used for estimation, benchmarking, project management, infrastructure planning, bid planning, outsources management, standards compliance and budget support.

The data set is essentially a 250*40 matrix i.e. there is a record of values of 40 attributes for 250 projects. 70% of the data i.e. 175 records are considered as the training data set and the remaining 75 records are used for Testing Data-set.

The training data-set is used to extract principal components for factor analysis. And the estimation process is run on the training dataset. Then the results of estimation process are compared with the actual cost values so as to determine the accuracy of estimates.

7.3 Software/tool used:

- For Principal component analysis of the agile project data the Statistical Toolbox of MATLAB software is used.
- For Constraint Programming the Oz constraint programming language is used. Oz is a multi-paradigm programming language allows to predefine the set of constrains that must be satisfied constantly, rather than to write methods to maintain the relations. The Mozart Compiler called "Mozart Programming System" is the primary implementation of Oz. It is released with an open source license for various platforms such as Unix, FreeBSD, Linux, Microsoft Windows, and Mac OS X.

7.4 Steps of Cost Estimation Process and their results

The Principal Components are identified using the MATLAB software through accumulation contribution, communalities and scree plot as seen in previous chapter.
From the processed training data set we obtain the component correlation matrix.

The Component Correlation matrix contains the Factor loadings for each component which explains the co-relation between the corresponding component and the cost of development.

The following table shows these factor loadings for each of the 40 components).

Table 10. Factor loadings and total variance of all the components

S.No.	COMPONENT	EIGEN VALUE	% OF VARIANCE	CUMMULATIVE
1	Cumulative Function points	5.721	12.099	12.099
2	Team Size	4.451	9.414	21.513
3	Development Platform	3.465	8.328	29.841
4	Project complexity	2.904	7.142	36.983
5	Operating System	1.986	6.300	43.183
6	CMMI	1.899	6.016	49.199
7	Architecture	1.585	5.352	54.552
8	ISO	1.542	5.261	60.813

9	Application Type	1.313	5.177	66.590
10	Process maturity	1.242	4.627	71.216
11	Programming Language	1.019	4.155	75.372
12	Risk taking	1.001	4.007	79.203
13	Early delivery	0.865	3.829	82.318
14	Software Size	0.851	1.900	83.118
15	Managerial skills	0.837	0.870	84.888
16	Technical ability	0.824	0.843	85.631
17	Planning	0.782	0.754	86.285
18	Data Base System	0.774	0.637	87.022
19	Reliability	0.756	0.599	87.920
20	Productivity	0.756	0.579	88.119
21	Following process model	0.746	0.478	89.697
22	Communication skills	0.745	0.476	90.273
23	Feedback	0.739	0.463	90.836
24	Tool availability	0.728	0.440	91.375
25	Organization Type	0.724	0.431	91.906
26	IDE	0.719	0.421	92.427
27	Tool familiarity	0.702	0.385	92.912
28	Proximity of team	0.694	0.368	93.380
29	Language Type	0.688	0.355	93.735
30	Hardware	0.671	0.349	94.254
31	Type of Server	0.658	0.325	95.645
32	Package Customization	0.652	0.319	96.024
33	Client/Server	0.617	0.305	96.329
34	Debugging capability	0.612	0.294	96.624
35	Training	0.581	0.229	97.852
36	Ease of use	0.578	0.222	98.075

37	Documentation period	0.561	0.186	98.261
38	Pages of documents	0.559	0.182	98.443
39	Documentation resources	0.534	0.129	99.917
40	Other Expenditure	0.512	0.083	100

The following table shows the Communalities extraction for each component.

Table 11. Communalities

S.No.	COMPONENT	EXTRACTION VALUE
1	Cumulative Function points	8.18246
2	Team Size	4.95285
3	Development Platform	3.001556
4	Project complexity	2.108304
5	Operating System	0.986049
6	CMMI	0.90155
7	Architecture	0.628056
8	ISO	0.594441
9	Application Type	0.430992
10	Process maturity	0.385641
11	Programming Language	0.25959
12	Risk taking	0.2505
13	Early delivery	0.187056
14	Software Size	0.18105
15	Managerial skills	0.175142
16	Technical ability	0.169744
17	Planning	0.152881
18	Data Base System	0.149769
19	Reliability	0.142884
20	Productivity	0.142884

21	Following process model	0.139129
22	Communication skills	0.138756
23	Feedback	0.13653
24	Tool availability	0.132496
25	Organization Type	0.131044
26	IDE	0.12924
27	Tool familiarity	0.123201
28	Proximity of team	0.120409
29	Language Type	0.118336
30	Hardware	0.11256
31	Type of Server	0.108241
32	Package Customization	0.106276
33	Client/Server	0.095172
34	Debugging capability	0.093636
35	Training	0.08439
36	Ease of use	0.083521
37	Documentation period	0.07868
38	Pages of documents	0.07812
39	Documentation resources	0.071289
40	Other Expenditure	0.065536

From the above two tables we extract 12 factors which have significantly high extraction values. The Cumulative of first 12 components is 0.792. This means that the first 12 components explain 79.2% of the variation in the Development Cost. Hence we will consider the first 12 components as the **Principal Components**. The eigen value for the rest of the components is below 1. The same can also be seen from the scree plot in figure18.

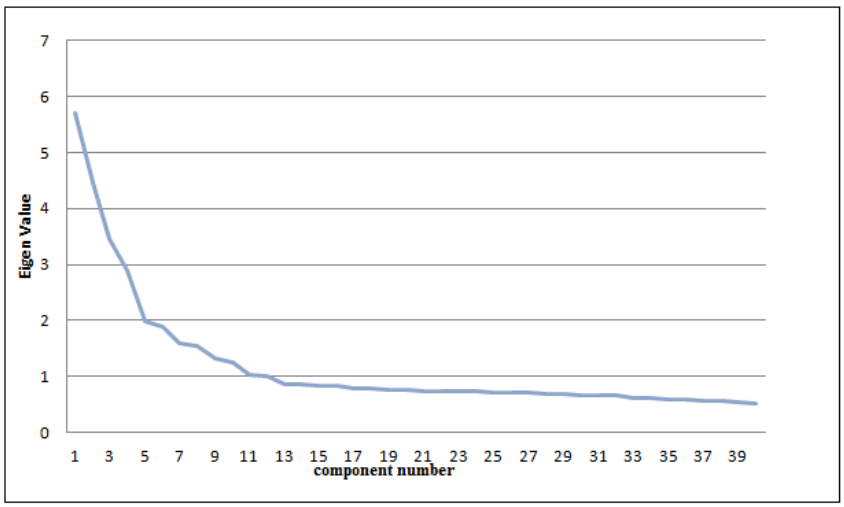

Fig 18. Scree plot

We can see that the line becomes nearly flat and below 1 after the first 12 components. Hence the extracted factors using Principal Component Analysis are:

Table 12. cost estimation factors

Cumulative Function points	Architecture
Team Size	ISO
Development Platform	Application Type
Project complexity	Process maturity
Operating System	Programming Language
CMMI	Risk taking

For the extracted factors a Principal Component Coefficients matrix is generated. It describes the relative scores of the principle components for the purpose of Cost Estimation.

Table 13. Component Coefficient matrix

COMPONENT Number	COMPONENT	COEFFICIENT VALUE
1	Cumulative Function points	0.274
2	Team Size	0.094
3	Development Platform	0.054

4	Project complexity	0.646
5	Operating System	0.096
6	CMMI	0.705
7	Architecture	0.419
8	ISO	0.541
9	Application Type	0.012
10	Process maturity	0.385
11	Programming Language	0.259
12	Risk taking	0.255

On obtaining the set of principal components/attributes and the score coefficients matrix we have to determine the criteria set required to emulate the agile development environment. The OZ implementation code for this is given in Annex-1.

The Input is the Component Coefficient matrix as obtained in previous step. This code is run on the Test data-set while specifying the constraints respective to each manifest.It solves the given constraints and outputs the development cost factor for each record in the test data-set.

The estimated cost value and actual cost value for each of the project as given in the testing data set is given as follows:

Table 14. results of estimation process

Record No.	**Actual Development Cost**	**Estimated Development Cost**
1	2040	1027.997
2	1600	1009.334
3	243	166.652
4	240	355.107
5	33	45.595
6	43	22.587
7	80	9.774
8	1075	452.773
9	423	229.127
10	321	189.945
11	218	170.653
12	201	102.792
13	79	170.108

14	60	30.157
15	61	100.652
16	40	70.741
17	9	23.917
18	11400	3469.998
19	6600	1820.674
20	6400	3096.462
21	2455	923.864
22	724	453.347
23	539	270.614
24	453	241.959
25	523	175.878
26	387	182.817
27	88	57.906
28	98	232.928
29	7	27.781
30	5	6.519
31	1063	747.700
32	702	957.050
33	605	359.256
34	230	121.408
35	82	98.398
36	55	35.076
37	47	87.706
38	12	19.719
39	8	18.427
40	8	19.786
41	6	14.690
42	45	109.272
43	83	103.778
44	87	132.802
45	106	109.269
46	126	213.934
47	36	32.778
48	1272	1830.003
49	156	110.712
50	176	115.826
51	122	82.728
52	41	40.164
53	14	21.255
54	20	11.771
55	18	7.515

56	958	271.801
57	237	207.013
58	130	102.387
59	70	82.787
60	57	50.117
61	50	47.269
62	38	7.641
63	15	13.863
64	278	297.900
65	656	698.008
66	67	36.780
67	678	622.054
68	34	85.012
69	677	477.987
70	890	687.456
71	346	268.897
72	456	226.897
73	23	9.126
74	76	89.697
75	89	103.375

In the next section we will compare the estimated costs with the actual values.

7.5 Evaluation of the Estimation Process

The proposed two step cost estimation model predicted the development cost for the given data-set. The estimated values were then compared against the Actual cost values.

This was done using MMRE (Mean Magnitude of Relative Error) .

MMRE values are computed from the relative error, or RE, which is the relative size of the difference between the actual and estimated value. It is calculated as follows:

$$RE_i = (estimate_i - actual_i) / (actual_i)$$

$$MRE_i = abs(RE_i)$$

$$MMRE = (100/N)*(MRE_1 + MRE_2 + ... + MRE_N)$$

For the above results we derive that the

MMRE = 50.63

The below plot shows the trend of MRE of the estimate cost with respect to rising project cost. We can see that

- The error value is localised within the a small range around the mean value with the exception of a few projects.
- Increased cost of development has no effect on the estimation error value.
- This means that even for very complex and costly projects the error in estimation will be the same.

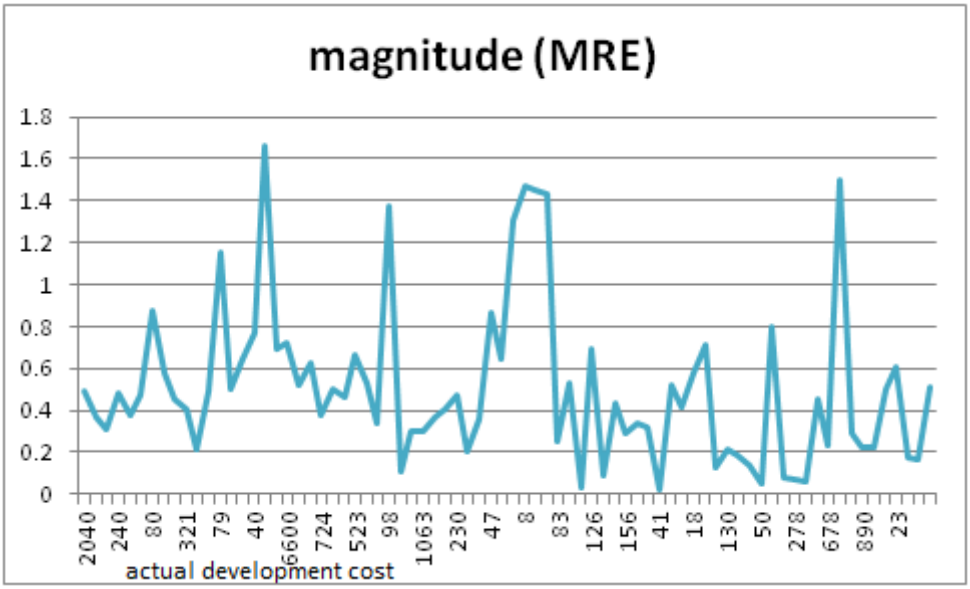

Fig 19.MRE vs development cost for proposed estimation model

7.6 Comparison with previously existing estimation approaches

In this section we will compare the proposed model with the models which are currently under research as studied in the chapter 5

We use the model-1 on our dataset. The results of estimation process are as shown in Table 15.

Table 15. results of Model-1 estimation process

Record No.	Actual Cost	Estimated Cost (EBV)
1	2040	5577.985
2	1600	4242.752
3	243	387.068
4	240	375.514
5	33	36.378
6	43	48.899
7	80	99.708
8	1075	2685.521
9	423	743.072
10	321	530.819
11	218	324.331
12	201	294.371
13	79	97.535
14	60	71.03
15	61	72.774
16	40	44.977
17	9	9.655
18	11400	35431.885
19	6600	19850.663
20	6400	18637.265
21	2455	6925.608
22	724	1616.358
23	539	1034.74
24	453	812.911
25	523	981.118
26	387	665.888
27	88	114.351
28	98	130.353
29	7	7.44
30	5	5.285
31	1063	2579.708
32	702	1526.04
33	605	1189.265
34	230	347.793
35	82	103.211
36	55	64.171
37	47	54.108
38	12	12.917

Record No.	Actual Cost	Estimated Cost (EBV)
39	8	8.528
40	8	8.554
41	6	6.359
42	45	51.482
43	83	105.543
44	87	111.811
45	106	142.744
46	126	174.154
47	36	40.063
48	1272	3272.941
49	156	221.73
50	176	253.864
51	122	166.405
52	41	46.356
53	14	15.123
54	20	21.857
55	18	19.592
56	958	2259.802
57	237	364.447
58	130	182.166
59	70	84.896
60	57	66.979
61	50	57.94
62	38	42.503
63	15	16.263
64	278	451.059
65	656	1321.192
66	67	80.579
67	678	1435.97
68	34	37.654
69	677	1397.811
70	890	2041.808
71	346	583.468
72	456	836.409
73	23	25.242
74	76	92.982
75	89	116.983

The plot of MRE (magnitude of relative error) is given as follows:

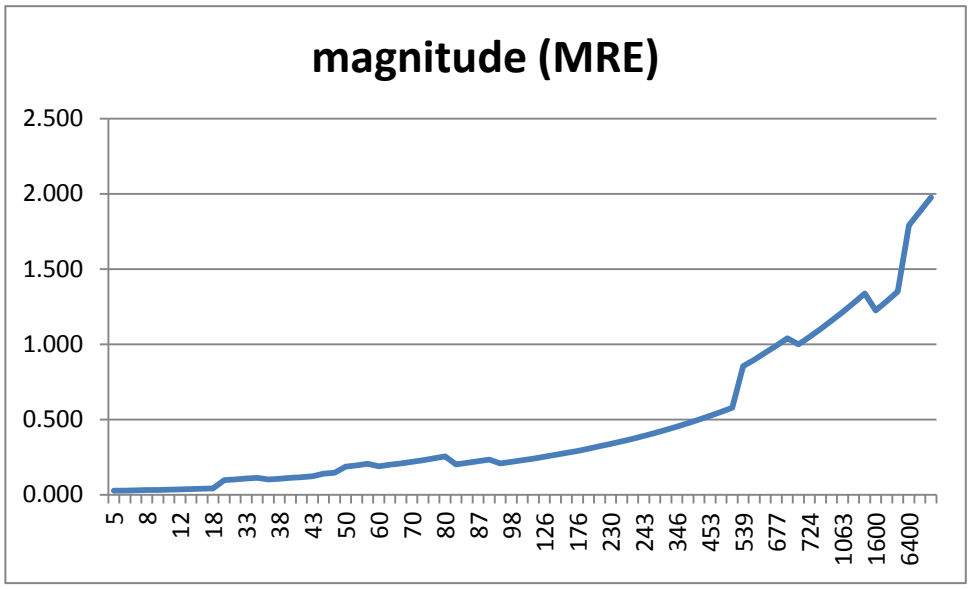

Fig20. MRE vs Actual development cost plot for model-1

From the above results and plot we see that

- The mean magnitude of relative error MMRE = 46.7
- The error value is affected by the increasing development cost of the project.
- This means that even though the MMRE value is less than our approach by a 4%, the actual value of the error in estimate is high for projects which are not small or low-budget. Hence our approach using PCA and constraint programming will be more suited to real-world Agile software projects.

Next we use the model-2 estimation approach. The results of estimation process are as follows:

Table 16. results of Model-2 estimation process

Record No.	Actual Cost (EBV)	Estimated Cost (EBV)
1	2040	1027.997
2	1600	1009.334
3	243	166.652
4	240	355.107
5	33	45.595
6	43	22.587

Record No.	Actual Cost (EBV)	Estimated Cost (EBV)
39	8	18.427
40	8	19.786
41	6	14.690
42	45	109.272
43	83	103.778
44	87	132.802

7	80	9.774
8	1075	452.773
9	423	229.127
10	321	189.945
11	218	170.653
12	201	102.792
13	79	170.108
14	60	30.157
15	61	100.652
16	40	70.741
17	9	23.917
18	11400	3469.998
19	6600	1820.674
20	6400	3096.462
21	2455	923.864
22	724	453.347
23	539	270.614
24	453	241.959
25	523	175.878
26	387	182.817
27	88	57.906
28	98	232.928
29	7	27.781
30	5	6.519
31	1063	747.700
32	702	957.050
33	605	359.256
34	230	121.408
35	82	98.398
36	55	35.076
37	47	87.706
38	12	19.719

45	106	109.269
46	126	213.934
47	36	32.778
48	1272	1830.003
49	156	110.712
50	176	115.826
51	122	82.728
52	41	40.164
53	14	21.255
54	20	11.771
55	18	7.515
56	958	271.801
57	237	207.013
58	130	102.387
59	70	82.787
60	57	50.117
61	50	47.269
62	38	7.641
63	15	13.863
64	278	297.900
65	656	698.008
66	67	36.780
67	678	622.054
68	34	85.012
69	677	477.987
70	890	687.456
71	346	268.897
72	456	226.897
73	23	9.126
74	76	89.697
75	89	103.375

The plot of MRE (magnitude of relative error) is given as follows:

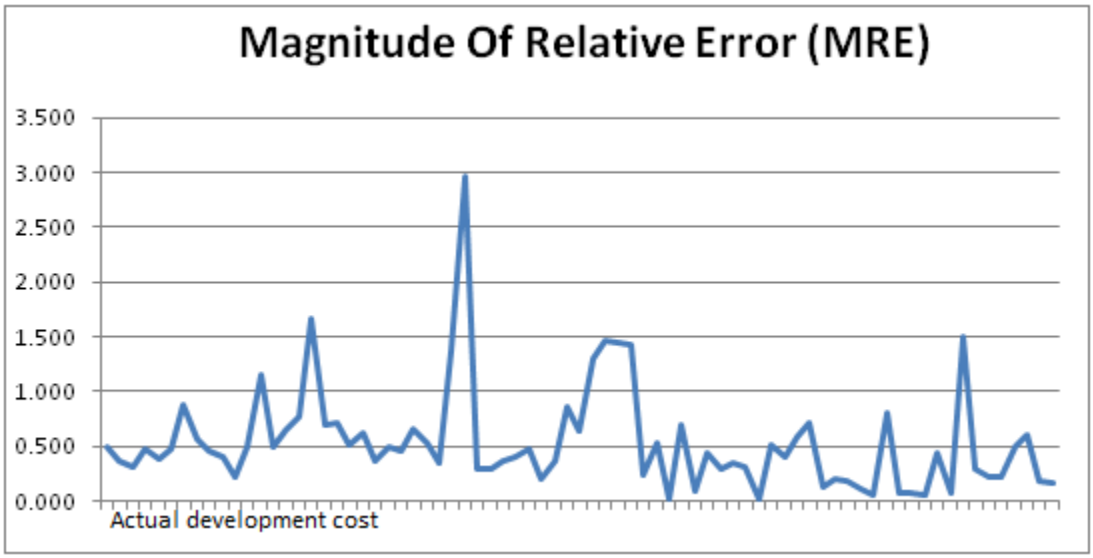

Fig21. MRE vs actual development cost plot formodel-2

From the above results and plot we see that

- The mean magnitude of relative error MMRE = 64.37
- This is higher than the results of our PCA based approach. Hence our approach using PCA and constraint programming will be more suited to real-world Agile software projects.

7.7 RESULTS

Results can be stated as follows:

- We have extracted 12 factors (or components) which account for the most variation (79.2%) in the Development Cost. Thus we can discard the remaining attributes which estimating development cost in the project planning stage.

- The proposed methodology is most suitable for agile projects as it uses constraint programming to explicitly check for satisfaction of agile manifestos. Each of the extracted factor is associated with one of the 4 manifestos and the constraint Programming implementation (using OZ) is done so that the manifesto criteria are met.

- The proposed methodology can be used in case of unavailability of historical data or expert opinion. Most Agile software Cost estimation processes rely on expert opinion or planning poker based cost estimation methods. However in case that is not available then the extracted 12 factors can be used by the development team to estimate software development cost while still satisfying the conditions of agile manifesto.

- On comparison with other approaches under research we find that our model provides a low MMRE value i.e.50.63; which is marginally lower than that seen with Planning Poker i.e. 106.81[24]. Also the estimation error does not increase in high cost/complexity projects. Hence we can safely say that the proposed cost estimation approach increases the precision and accuracy of estimates; and hence is better suited for the Agile Software Development Projects.

CONCLUSION

- Most of the existing cost estimation techniques have been developed to support traditional sequential software development methodologies whereas Agile Software Development is iterative in nature. If these traditional techniques are used for estimation in Agile software projects, then the results will be definitely inaccurate.
- Only COSMIC published a measurement guideline for ASD methods and in this guideline only software size will be estimated. Yet there is no well-defined and standard other measurement practices in ASD methods.
- It has been observed that agile methods mostly rely on an expert opinion and historical data of project for estimation of cost, size and duration. It has been observed that these methods do not consider the vital factors affecting the cost, size and duration of project for estimation. In absence of historical data and experts, existing agile estimation methods such as analogy, planning poker become unpredictable. Therefore, there is a strong need to devise simple algorithmic method that incorporates the factors affecting the cost, size and duration of project. It will also provide the basis for inexperienced practitioners to estimate more precisely.
- The proposed cost estimation model does just that. In this estimation approach we have extracted 12 agile development factors which affect Development Cost. Principal Component Analysis Technique is used to extract these factors.

  The resulting factors along with their coefficients are used to predict software development cost constraint programming technique. This estimation is modelled through a careful consideration of all attributes relating to the four agile manifestos.

  This cost estimation model is suitable for software development teams working in agile environment

  Future work will focus on finding more cost affecting attributes for different categories of agile software development projects so as to improve the precision of the model. Secondly we can in-corporate the similarity difference measurement while assigning coefficients/weights to the factors.

References

[1] K. Beck, A. Cockburn, R. Jeffries, And J. Highsmith, Agile Manifesto,2001, http://www.ggilemanifesto.Org.

[2] Pressman R. S.,'Software Engineering: A Practitioner's Approach', Mcgraw-Hill, 1997

[3] Agile Software Development, Alistair Cockburn, Addison Wesley Professional, 2002

[4] M. Cristal, D. Wildt And R. Prikladnicki, 'Usage Of Scrum Practices Within A Global Company', IEEE International Conference On Global Software Engineering (ICGSE 2008), Pp222- 226, IEEE, 2008

[5] M. Singh, 'U-Scrum: An Agile Methodology For Promoting Usability In Agile', Agile '08 Conference, Toronto, 2008.

[6] Ceschi M., Sillitti A., Succi G. & De Panfilis S, 'Project Management In Plan- Based And Agile Companies', IEEE Software, Vol22, Pp21-25,2005

[7] Lederer A. L. & Prasad J, 'Perceptual Congruence and Information Systems Cost Estimating', 1995 ACM Sigcpr, Nashville, Tennessee.

[8] Ewusi-Mensah, K. & Przasnyski, Z. H. ,'Learning From Abandoned Information Systems Development Projects', Journal Of Information Technology, Vol10,Pp 3-14,1995

[9] Schalliol G., 'Challenges For Analysts on A Large XP Project', XP Universe 2001, Raleigh, North Carolina.

[10] Jørgensen, M. & Moløkken, K., 'A Preliminary Checklist For Software Cost management', Proceedings Of The 3rd International Conference On Quality Software, 2003

[11] Stamelos, I. & Angelis, L., 'Managing Uncertainty in Project Portfolio Cost Estimation', Information and Software Technology, Vol43, Pp759-768, 2001

[12] Strike, K., El Emam, K. & Madhavji, N., 'Software Cost Estimation with Incomplete Data', IEEE Transactions on Software Engineering, Vol27, Pp890-908, 2001

[13] Jørgensen, 'A Review of Studies on Expert Estimation of Software Development Effort', Journal of Systems and Software, 2004

[14] Jørgensen, 'Top-Down And Bottom-Up Expert Estimation of Software Development Effort', Information and Software Technology, 2004

[15] Boehm, B. W., Abts, C. & Chulani, 'Software Development Cost Estimation Approaches: A Survey', Usc-Cse, 2000

[16] Abrahamsson, Pekka, Et Al. 'Predicting Development Effort from User Stories', 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2011.

[17] Agarwal, R., Kumar, M., Yogesh, Mallick, S., Bharadwaj, R. M. & Anantwar, D. ,'Estimating Software Projects. ACM Sigsoft Software Engineering Notes, Vol26, Pp60-67, 2011

[18] Finnie, G. R., Wittig, G. E. & Desharnais, J.-M., 'A Comparison Of Software Effort Estimation Techniques: Using Function Points With Neural Networks, Case-Based Reasoning And Regression Models', Journal Of Systems And Software, Vol 39, Pp 281-289, 1997

[19] Jørgensen, M., Indahl, U. & Sjøberg D., 'Software Effort Estimation By Analogy And Regression Toward The Mean', Journal Of Systems And Software, Vol 68, Pp253-262, 2003

[20] Asnawi, Ani Liza, Andrew M. Gravell, and Gary B. Wills, 'Factor Analysis: Investigating Important Aspects for Agile Adoption', Agile India 2012, IEEE, 2012.

[21] Santos Ma, Bermejo Phs, Oliveira Ms, Tonelli Ao, 'Agile Practices: An Assessment Of Perception Of Value Of Professionals On The Quality Criteria In Performance Of Projects', Journal Of Software Engineering And Applications, V. 04, P. 700-709,2011

[22]Moser Raimund, Witold Pedrycz, and Giancarlo Succi, 'A Comparative Analysis Of The Efficiency Of Change Metrics And Static Code Attributes For Defect Prediction', ACM/IEEE 30th International Conference On Software Engineering, 2008 (Icse'08), IEEE, 2008

[23] M. Cohn, User Stories Applied: For Agile Software Development, 1 Ed.: Addison-Wesley, 2004.

[24] Haugen, N., 'An Empirical Study of Using Planning Poker for User Story Estimation', Proceedings Of Agile 2006 Conference, 2006

[25] Moløkken-Østvold, Kjetil, Nils Christian Haugen, And Hans Christian Benestad, 'Using Planning Poker For Combining Expert Estimates In Software Projects', Journal Of Systems And Software 81.12, Pp 2106-2117, 2008

[26] J.-M. Desharnais And L. Buglione, 'Using The Cosmic Method To Estimate Agile User Stories', Presented At The Proceedings Of The 12th International Conference On Product Focused Software Development And Process Improvement, Torre Canne, Brindisi, Italy, 2011.

[27] Cosmic, "Guideline For The Use Of Cosmic FSM To Manage Agile Projects," Ed, 2011.

[28] Ziauddin, Shahid Kamal Tipu, Khairuz Zaman, And Shahrukh Zia, 'An Effort Estimation Model For Agile Software Development', Advances In Computer Science And Its Applications (ACSA) Vol 2 (2012): 314-324, 2012.

[29] Tessem, Bjørnar, 'Experiences in Learning XP Practices: A Qualitative Study.' Extreme Programming and Agile Processes in Software Engineering, Springer Berlin Heidelberg, 131-137, 2003

[30] Lee G, Xia W, 'Toward Agile: An Integrated Analysis of Quantitative And Qualitative Field Data On Software Development Agility', Communications Of The ACM 47.5 (2004), 68-74

[31] Capers Jones,"Estimating Software Costs: Bringing Realism to Estimating",2007, Tata McGraw-Hill , Second Edition.

[32] A. Tosun, B. Turhan, and A. B. Bener, "Feature weighting heuristics for analogy-based effort estimation models," Expert Systems with Applications, vol. 36, no. 7, pp. 10 325–10 333,Sep. 2009.

[33] G Mathur, K Jugdev, S Fung, "Project management assets and project management performance outcomes- Exploratory factor analysis", Management Research Review, Vol.36 No.2, 2013

[34] J Weng, Shixian Li, Linyang Tang, "Improve Analogy-Based Software Effort Estimation Using Principal Components Analysis And Correlation Weights", 16th Asia-Pacific Software Engineering Conference, IEEE, 2009

[35]Krohn, David, Daniel Marino-Johnson, and John Paul Ouyang. "The KMO Method for Solving Non-homogenous, m th Order Differential Equations." Rose-Hulman Undergraduate Mathematics Journal 15.1 (2014)