Chapter 1

Introduction

It is never possible to construct a strong house without a strong foundation and a good architect. Similarly, in order to design and develop a reliable and usable software product that meets all the requirements of the stakeholders; a suitable development lifecycle is necessary. Selection of appropriate software development lifecycle is very essential for failure free development of software projects. It is very difficult to backtrack and start with new lifecycle especially, if the methodology chosen for the project is not appropriate. The selection of lifecycle is very involved task. It is not easy to select a software development lifecycle for a project on the basis of few characteristics. So, a large number of metrics are considered and each metric is assigned an appropriate weight. On the basis of these weighted metrics and the value of metrics for desired project, we are able to identify the appropriate methodology for the project to be developed.

Various lifecycle methodologies has been proposed such as traditional methodology, object oriented methodology and agile methodology. Some of the traditional methodology includes waterfall model, spiral model, prototype model and many more. The major problem with the waterfall model is its inflexibility, which makes it difficult to respond to changing customer requirements. Therefore, the waterfall model should only be used when the requirements are well understood and unlikely to change during development. Spiral model has gained popularity over other traditional approach. The popularity of spiral model is due to its high sensitivity to risk at each stage of development, wide range of options to accommodate the good features of other lifecycle models. Spiral model is appropriate for large-scale enterprise systems. Several object-oriented design methods have been proposed (Coad and Yourdon,

1990; Robinson, 1992; Jacobson, et al., 1993; Graham, 1994; Booch, 1994).The UML (Rumbaugh, et al., 1999) is a unification of the notations used in these methods.

The RUP recognizes that conventional process models present a single view of the process[15].

RUP is described from three perspectives:-

1. A dynamic perspective that shows that phases of the model over time.
2. A static perspective that shows the process activities that are enacted.
3. A practice perspective that suggests good practices to be used during the process.

These approaches involve a significant overhead in planning, designing and documenting the system. When this heavyweight, plan-based development approach was applied to small and medium-sized business systems, the overhead involved was so large that it sometimes dominated the software development process. More time was spent on how the system should be developed than on program development [15]. Dissatisfaction with these heavyweight approaches led a number of software developers in 1990s to propose new agile methods. **These allowed the development team to focus on the software itself rather than on its design and documentation. Agile methods are intended to deliver working software quickly to customers and allow changes in requirements to be included in later iteration of the system.**

Well known agile approaches include Extreme Programming, Scrum, Crystal, Adaptive Software development, DSDM and Feature Driven Development. Extreme Programming (XP) is perhaps the best known and most widely used of the agile methods. This approach was developed by pushing recognized good practice, such as iterative development, and customer involvement to ‘extreme’ levels [15].

1.1 Motivation

A large number of projects fail during their development phases due to inappropriate lifecycle selection. It is not feasible to start with a randomly chosen software development lifecycle methodology. For successful completion of project in budget and target time, methodology plays an important role.

Today's market is global market. That is, business area is not limited to a state or country but it operates over the world. Therefore, businesses have to respond to new opportunities and markets, changing economic conditions and the emergence of competing products and services. Software is involved in almost all business operations so, its development should be fast enough to take advantage of new opportunities and to respond to competitive pressure.

Therefore, the most critical requirement for software system is rapid development and delivery. In fact, at initial stages, many businesses compromise on software quality and requirements against rapid software delivery. Because of the challenging and volatile market it is practically impossible to derive a complete set of requirements. The real requirements become clear only after a system has been delivered and user gain experience with it. Software development process that are based on completely specifying the requirements then designing, building and testing the system are not suitable for rapid software development. As the requirements change or as requirements problems are discovered, the system design or implementation has to be reworked and retested. As a consequence, a conventional waterfall or specification-based process is usually prolonged and the final software is delivered to the customer long after it was originally specified.

In a fast-moving business environment, this can cause real problems. By the time the software is available for use, the original reason for its procurement may have changed so radically that the software is effectively useless. Therefore, for business systems in particular, development processes that focus on rapid software development and delivery are essential.

All methods have limits, and agile methods are only suitable for some types of system development. They are best suited to the development of small or medium-sized business system and personal computer products. They are not well suited to large-scale systems development with the development teams in different places and where there may be complex interactions with other hardware and software systems. Nor should agile methods be used for critical systems development where a detailed analysis of all of the system requirements is necessary to understand their safety or security implications.

Therefore, selection of methodology should be supported by a framework which will guide you to select an essential project specific methodology.

Software development lifecycle selection depends on multiple factors of the project. We have discussed the lifecycle selection based on requirements, development team, users, project type and associated risk.

1.2 Related work

Methods make the software development task easy, efficient, systematic and resourceful. But it is fact that there is no universal method that can be applied to all projects since different projects have different characteristics and situations. This requirement creates home ground for Method Engineering (ME). Method Engineering has gained popularity with the first widely accepted definition by Brinkkemper. He has defined ME as a "discipline to design, construct and adapt methods, techniques and tools for an Information System Domain (ISD) project" [3].

There are number of proposals for developing project specific methods. They can be broadly divided into two classes: (i) method assembly which relies on method base which stores the method component, an atomic element of a method. From this method base method components are retrieved as per the project requirements .The retrieved components are then assembled to form situation specific method [23]. Some important proposals are Fragment

base approach [9, 10], GOPRR approach [19] and Contextual approach [28]. (ii) Instantiation approach where project specific method is built from scratch Building the method from scratch. All these approaches require instantiation of a Meta-model where; the concepts of a method are made instances of meta-model concepts. Since instantiation is a tedious and time-consuming task, Gupta [8] had proposed a method requirement specification language rooted in a simple meta-model i.e. Method View Model (MVM) having only limited concepts, mitigated the problem of instantiation to a great extent. All these approaches were supported by computer based tool support.

The task of method engineers is complex in nature. In order to facilitate them there are proposals to provide a rich set of rules and guidelines to form a coherent method[15] .These proposal are analogous to architectural- based software engineering domain proposals. They provide for the task to be performed in a more disciplined and cohesive way. The major ones are Method Intension Architecture (MIA) and Architectural Centric Method Engineering (ArCME). In these approaches there are problems like suitable style selection and further composition of these selected styles. OPF [32] solves these issues due to their flexible nature but fails to address a wide variety of concepts like branching, situation cataloguing and evolution tracing [6].

Major limitations with these approaches are that they are centered around traditional method. Coherence of method is supported by meta model which can only model traditional method and not agile method. The field of method configuration is gaining popularity where new method is configured from the base method according to the requirement. The task of configurability is to first create a new model called a configurable model followed by selecting those parts of the configurable model that are relevant to the user's requirement. Configurable models use notions of commonality and variability[34]. Coplien et al [12] define **commonality** as an assumption held uniformly across a given set of objects whereas

variability is an assumption that is true for only some elements of the set. In [33] we have the definition of **variability** as "an assumption about how members of a family may differ from one another". A configurable model identifies commonality and variability that can be exploited in developing a new system from the configurable model.

1.3 Problem statement

Recent trend has changed and agile methodologies are gaining popularity due to its significance like customer involvement, early software release, less documentation required and customer satisfaction. Lifecycle selection depends on various characteristics of the project under consideration such as, complexity, risk involved, Programmer's Capability, Clarity and completeness of requirements, Business Risk, and many more. There are project like development of air traffic controller, missiles and safety system software where agile methodology is not suitable. Since the selection of methodology is to be done prior to development of the project, we must evaluate project characteristics like complexity, modularization of task, business risk, technical risk, and programmer's capability to decide whether to use agile or traditional methodology. Weight assigned to these project characteristics play a major role in this decision. Hence problem of the thesis is:

Framework for selection of suitable methodology based on evaluation of project characteristics.

1.4 Scope of Work

We experiment our proposal for four projects. These are mobile app development, air traffic controller, ERP implementation for SMEs and software for banking system. The first step should be identification of project characteristics. Some of the characteristics have more impact on lifecycle selection and some of them having less impact. So, the next step should be distribution of weight to the identified project characteristics. Initially, the weight has been distributed manually and result has been obtained for the example project. Since this process

is complex and lot of calculation is required. So, we need to apply the AI approach for calculation. Further, weight has been adjusted using neural network. The accuracy of our implementation mainly depends on the correctness of the data set.

Broadly, the scope of this work can be summarized as:

i. **Identification of project characteristics for lifecycle selection.**

ii. **Finding impact of each of the above identified characteristics on the lifecycle selection.**

iii. **Assigning weight to each of the characteristics based on their impact.**

iv. **Run the framework for example project and observe the output.**

v. **Apply neural network for selection of software development lifecycle.**

Initially for mobile domain we had taken 14 project characteristics to decide the suitable lifecycle. Further we have included 8 more characteristics and input range is also modified. Instead of three input ranges low, medium and high, we have used five input category very low, low, medium, high and very high. The insertion of more project metrics and enhancement in input ranges makes our decision support system more robust. In our previous work we have not applied any machine learning technique. This time we have used neural network for weight adjustment of each identified project characteristics.

1.5 Organization of the Thesis

Chapter 1 begins with introduction and further it discusses motivation of undertaking this research work, related work, problem statement, scope of work as well as organization of this dissertation.

Chapter 2 provides literature survey. The concept of traditional and agile software development lifecycle is described in this section. Various challenges associated with traditional and agile software development lifecycle is also discussed. We have discussed the lifecycle selection based on requirements, development team, users, project type and associated risk. Neural Network concept is also discussed in this section.

Chapter 3 begins with description of our research work. It describes the framework of our approach. We have explained various steps involved in our approach in detail with some examples. Here we have shown the outcomes of the experiments with the help of graphs for different examples. We have calculated sum of product of all the factors for each example. In the end output for all the examples are compared with the help of a graph.

Chapter 4 maps the model with neural network model. It provides the training details of the neural network and also provides outcome of neural network.

Chapter 5 gives the final findings and outcomes of the research. It lists the problems that we have solved and those that still remain to be tackled. It also lays the ground to the future work in this direction.

Chapter 6 gives the details of our accepted paper titled *"**Domain specific priority based implementation of mobile services- an agile way**"* in *International Conference on Software Engineering and Research Practices (SERP-12), Las Vegas, USA.*

Chapter 2

Literature Review

The objectives of this chapter are to introduce various software development lifecycle methodologies and to provide a framework for understanding the rest of the work.

What is software development lifecycle?

The IEEE standard glossary of software engineering defines software development lifecycle as "the period of time that starts when a software product is conceived and ends when the product is no longer available for use". The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase.

There are various lifecycle model exists that are used for the development of project. On a broad scale we divide these models in three categories that are:-

- Conventional Methodology
- Object Oriented Methodology
- Agile Methodology

Basically conventional methodology considers a systematic way of software development and prepares a huge list of documents that reflects the functionalities of the product and shows the procedure to be used for development. Agile methodology believes in little documentation and quick delivery of the most prioritized task to the customer based on their choice[2].

2.1 Well known conventional Software Development Lifecycle Models

1. Rapid application development model

This model was developed by IBM in 1980s. In this model user involvement is essential from requirement phase to delivery of the product. The process is started with building a rapid prototype and is given to user for evaluation. The user feedback is obtained and prototype is refined. The process continues, till the requirements are finalised.

Advantages of the Rapid application development model

- Quick initial views of the product are possible due to delivery of rapid prototype.
- Development time of the product may be reduced due to use of powerful development tools like CASE tools.
- Involvement of user may increase the acceptability of the product.

Disadvantages of the Rapid application development model

- Not an appropriate model in the absence of user throughout the lifecycle.
- Development time may not be reduced, if reusable components are not available.
- Highly specialized & skilled developers are required and such developers are not easily available.

2. Spiral model

Important software projects have failed because project risks were neglected and nobody was prepared when something unforeseen happened. Barry Boehm recognized this and tried to incorporate the project risk factor into a lifecycle model, which results in spiral model. The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360 degree represents one phase.

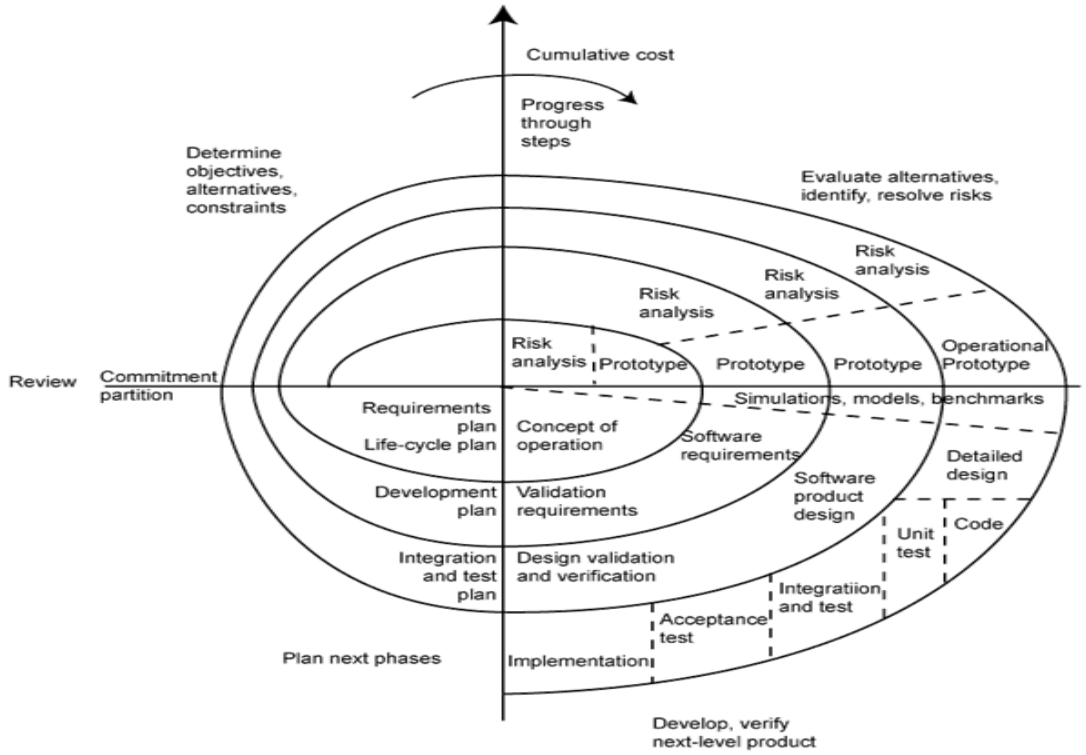


Fig. 1: Spiral Model

One phase is split roughly into four sectors of major activities.

Planning: Objectives are determined, planned and documented.

Risk Analysis: Analyse the risks and find alternatives and attempts to resolve the risks.

Development: Product development and testing product.

Assessment: Customer evaluation.

Advantages of the spiral model include the following:-

- Appropriateness for large-scale enterprise systems.
- Flexibility in terms of its sensitivity to the dynamic nature of the software industry.
- High sensitivity to risk at each stage of development.
- Wide range of options to accommodate the good features of other lifecycle models.

Disadvantages of the spiral model include the following:-

- Complex nature makes it difficult for customers to grasp.
- Requires extensive information regarding risk assessment.
- Undetected risks can be problematic.

3. Rational Unified Process model

The unified process is developed by I. Jaccobson, G. Booch and J. Rumbaugh. It is an iterative process model where project is developed through a series of short, fixed length mini projects called iterations. The outcome of each iteration is a tested, integrated and executable system[1]. Each iteration has its own requirement analysis, design, implementation and testing activities. Hence, the system continues to enlarge and refine with every iteration and thus grows incrementally over time[14].

There are four phases in the unified process that is shown in figure.

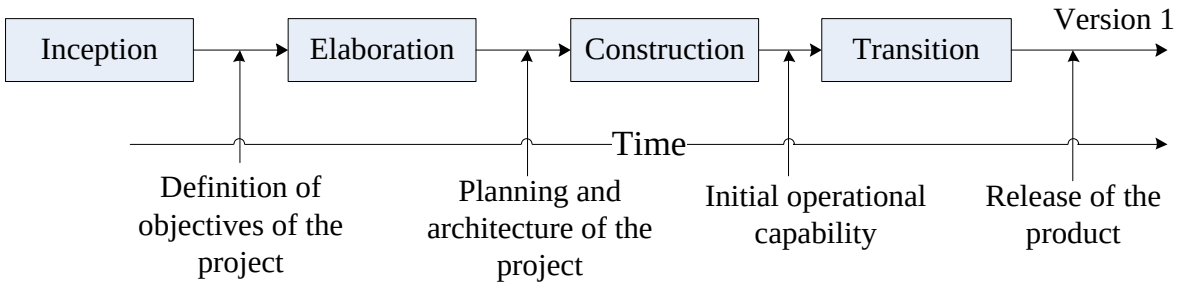


Fig.2:- Rational Unified Process Model

i. Inception: In this phase we define scope of the project.

ii. Elaboration: Here following things are explored about the project:-

- How do we plan & design the project?
- What resources are required?
- What type of architecture may be suitable?

iii. Construction: In this phase objectives are translated in design and architecture documents.

iv. Transition: Many activities takes place in this phase like delivering, training, supporting, and maintaining the product.

The RUP recognises that conventional process models present a single view of the process[15].

RUP is described from three perspectives:-

1. A dynamic perspective that shows that phases of the model over time.
2. A static perspective that shows the process activities that are enacted.
3. A practice perspective that suggests good practices to be used during the process.

Iterations and workflow of Rational Unified Process model:-

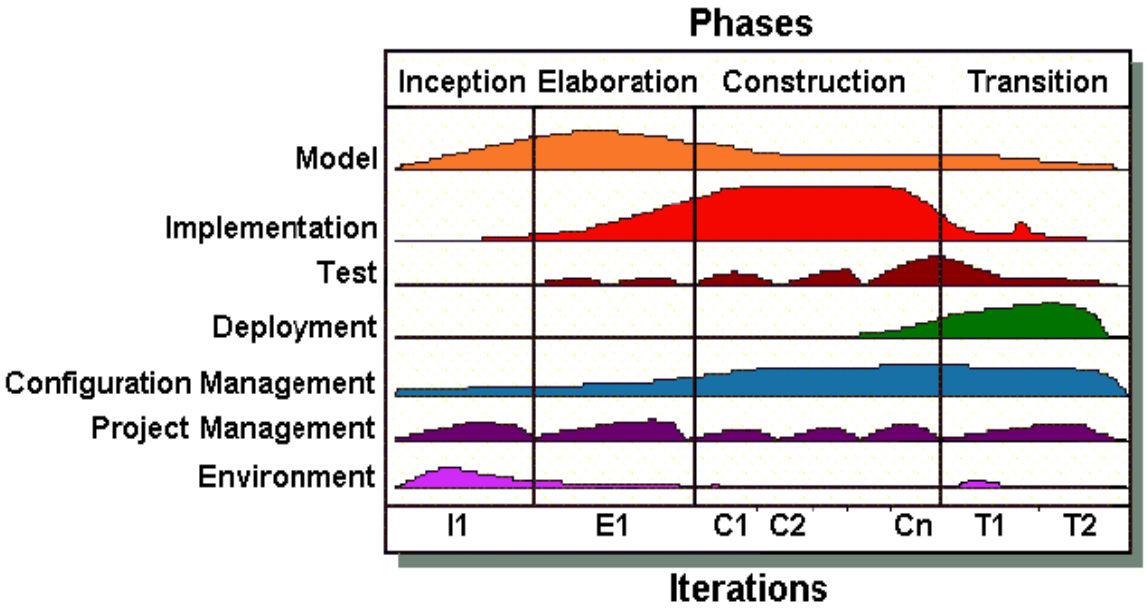


Fig.3:- Phase wise workflow of Rational Unified Process Model[14]

Advantages of Rational Unified Process model:-

- Changes can be accommodated in later stage of development.
- Risks can be minimised.
- We can reuse some version of the product because of iterative release.
- Multiple testing improves the quality of testing.

2.2 Well-known agile software development lifecycle models

In February 2001, 17 software developers published the *Manifesto for Agile Software Development* to define the approach now known as agile software development. Agile Manifesto reads, in its entirety, as follows:-

Individuals and interactions over processes and tools:- in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming[30].

Working software over comprehensive documentation:- working software will be more useful and welcome than just presenting documents to clients in meetings[30].

Customer collaboration over contract negotiation:- requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important[30].

Responding to change over following a plan:- agile development is focused on quick responses to change and continuous development. That is, while there is value in the items on the right, we value the items on the left more[30].

Agile Principles

Twelve principles underlie the Agile Manifesto, described below[31]:-

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation (co-location).

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity, the art of maximizing the amount of work not done, is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Agile methodology includes:-

- Scrum.
- Extreme Programming (XP).
- Feature Driven Development (FDD).
- Dynamic Systems Development Method (DSDM).
- Kanban.
- Lean Development.

Scrum

The scrum method is a general agile method that focuses on managing iterative development and does not adapt specific agile practices. It has three phases, they are discussed below:-

1. Outline planning phase for designing software architecture derived from general objective of the project. This is the management phase where first general objectives are elicited and these are then used to design software architecture.

2. Sprint cycle consisting of a product backlog which is converted into sprint backlog from where product is developed and delivered in increment. The cycle is repeated until the backlog is emptied.

3. Project closure phase which terminates the project and prepares the termination report and user manual.

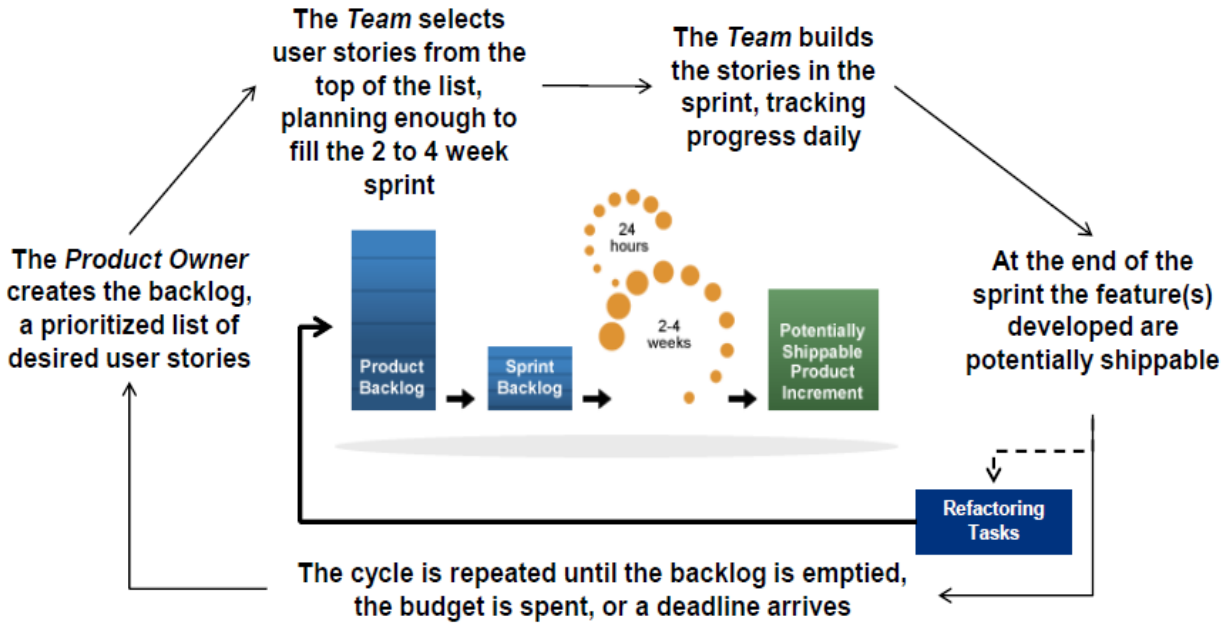


Fig.4 Understanding Scrum lifecycle model[39]

Once a shippable module is delivered, the product backlog is analyzed for the next priority module. Module reprioritization can also be done if required.

Example of Medical Diagnosis Software development using Scrum

Regular Services

1. Patient Registration(Details)
2. Search Appropriate Doctor on the basis of:-
    - Disease Type
    - Specialization of doctor
    - Experience of Doctor
3. Check Availability of Slot
4. Take appointment
5. Verify your Prescription(Valid dose etc)
6. Cancel appointment

Emergency Services

1. Check for availability of ICU wards
2. Register case
3. Admit Patient(allot ward, assign doctor)
4. Search Blood Bank
5. Check for availability of Blood
6. Order Blood

Table1:- Example of medical diagnosis software development using scrum

Product Backlog

Task	Priority
Check for availability of ICU wards	1
Register case	2
Admit Patient(allot ward, assign doctor)	3
Search Blood Bank	4
Check for availability of Blood	5
Order Blood	6
Patient Registration(Details)	7
Verify Prescription(Valid dose etc)	8
Search Appropriate Doctor	9
Check Availability	10
Take appointment	11
Cancel appointment	12

Sprint Backlog(Most Prioritized task to be completed in 1st sprint)

Check for availability of ICU wards
Register case
Admit Patient(allot ward, assign doctor)
Search Blood Bank
Check for availability of Blood
Order Blood

Extreme-Programming (XP)

Extreme Programming (XP) is very famous agile methodologies. It takes extreme approach to iterative development and delivers working software frequently.

In this, involvement of customer during development is very essential. The development team works in very close environment in presence of customer. There are six phases in XP that are discussed below.

1. Requirement Phase: - In this phase users give requirements as stories that are recorded on story cards then these story cards are prioritized.
2. Task gathering: - Here stories are broken into tasks.
3. Plan release phase: - In this phase the most prioritized stories are selected and planned for early release.

4. Development Phase: - The design of the only most prioritized task, which is to be developed, is done in this phase.

5. Release Phase: - The above designed task is developed and released for the use.

6. Evaluation of the system: - working of the released system is evaluated and the next cycle is started.

Example of user story for Medical Diagnosis Software

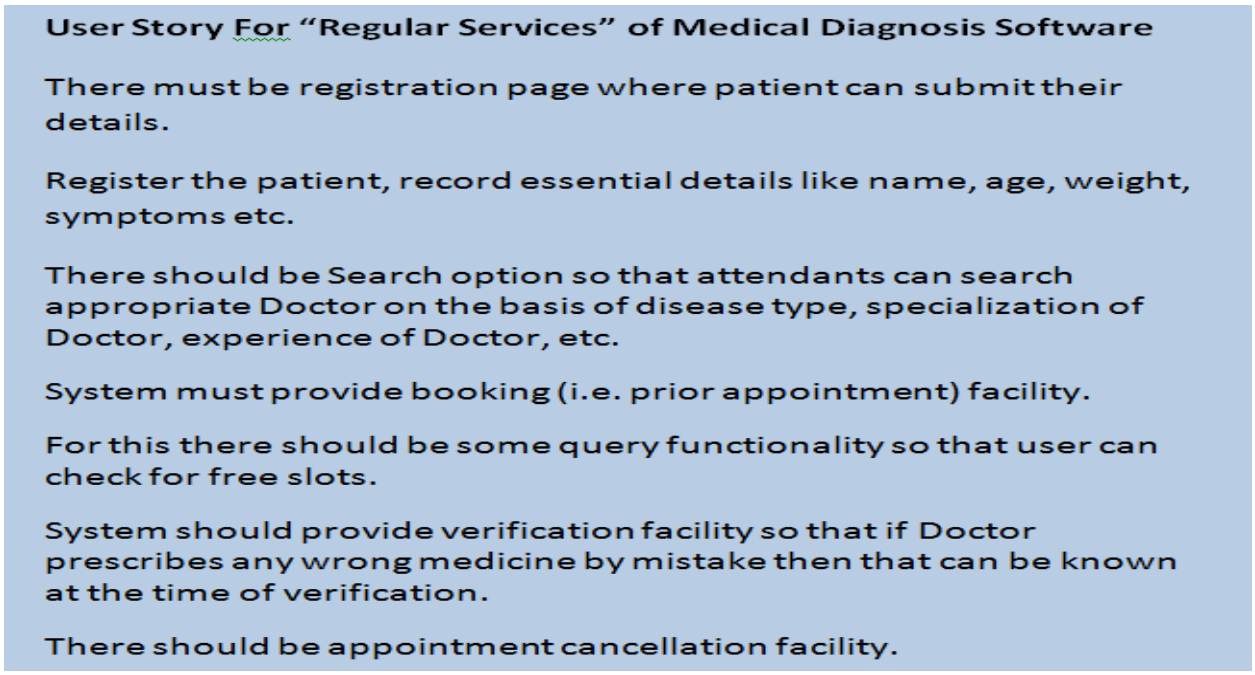


Fig.5:- Example showing user story for medical diagnosis software

StoryCard

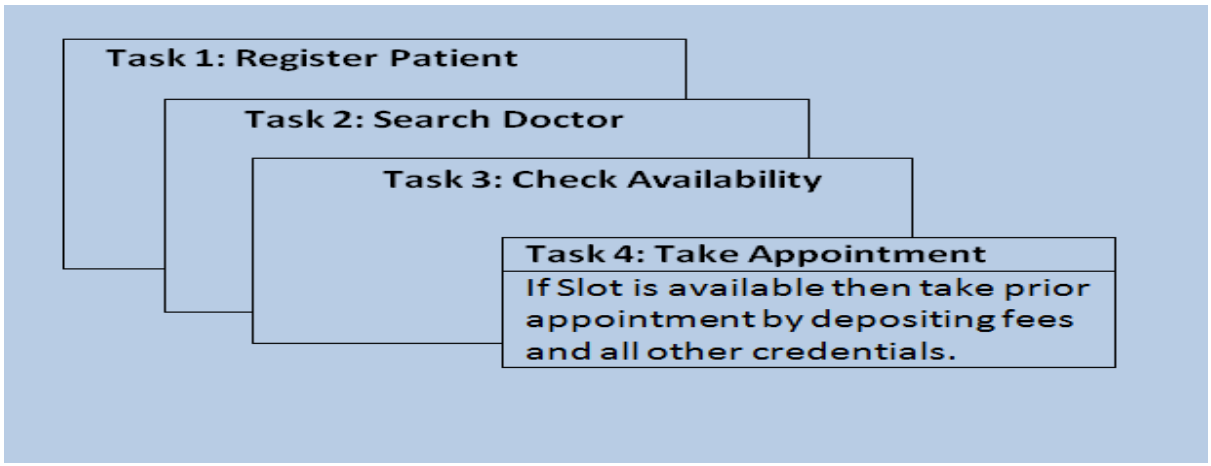


Fig.6:- Story card for the user story

Feature-Driven Development (FDD)

FDD is model driven methodology that releases software in form of features in short iterations. It is suitable for large team and consists of very short phases and delivers specific features in each phase.

Some of the Feature Sets for our project are "Incoming Call," "Outgoing Call," "Messaging," and this comes under "Basic Services" Subject Area. In FDD, we do planning, designing and building of a feature under consideration. It consists of five specific processes in specified order, which is discussed below.

1. Develop an overall model: - Requirements are gathered in top down approach where all subject areas are designed. Subject areas are aggregation of feature set. Feature set are combination of feature. Each feature is task to be performed.

2. Build a list of feature: - Features gathered are compiled to form feature list.

3. Plan by feature: - Planning is done to build a feature.

4. Design by feature: - Proper designing is done for the planned feature.

5. Build by feature: - Actual implementation of the feature is done.

Lean Development

Lean is very similar to Scrum in the sense that we focus on features as opposed to groups of features – however Lean takes this one step further again. In Lean Development, we select, plan develop, test and deploy one feature (in its simplest form) before we select, plan, develop, test and deploy the next feature. By doing this, we further isolate risk to a feature-level. In these environments, we aim to eliminate 'waste' wherever possible – we therefore do nothing until we know it's necessary or relevant.

2.3 Comparative Study of Traditional and Agile Methods

Different project requires different lifecycle models based on their characteristics. To decide, which lifecycle model is suitable for which type of project, it is essentially important to provide comparative study of both the models.

2.3.1 Waterfall

Waterfall model is a sequential approach to software development. A sequential approach means a stage by stage approach for software development. For example:-

- Analysis of business requirement by the project team is to be done in requirement analysis phase.
- Now the design of requirement collected are done, and a decision taken about which programming technique i.e. Java , Dot Net, etc. is to be used.
- After the design process, code implementation takes place.
- The testing of code is done in the next phase. That is the phase next to the coding phase is testing phase.
- Evaluation and maintenance of the product is done in the last phase, which ensures that everything runs smoothly.

Problems with waterfall model

- Difficulty in defining all requirements at the beginning of a project.
- Do not accommodate any changes at later stages.
- Working of the software is not visible until late in the project's life.
- This model is not appropriate for large projects.
- Real projects are rarely sequential.

2.3.2 Agile Development

It is a low over-head method that emphasizes values and principles rather than processes. Working in cycles i.e. a week, a month, etc., project priorities are re-evaluated and at the end of each cycle. Four principles that constitute Agile methods are:

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. And again, **responding to change** over plan follow-throughs.

The discussion below shows the reason for choosing Agile methodology over the Waterfall method.

1. Once a stage is completed in the Waterfall method, there is no going back, since most software designed and implemented under the waterfall method is difficult to change according to time and user needs. The problem can only be fixed by going back and designing an entirely new system, a very costly and inefficient method. Whereas, agile methods adapt to change, as at the end of each stage, the logical program, designed to cope and adapt to new ideas from the outset, allows changes to be made easily. With Agile, changes can be made if necessary without getting the entire program rewritten. This approach not only reduces overheads, it also helps in the upgrading of programs.

2. Another Agile method advantage is one has a launchable product at the end of each tested stage. This ensures bugs are caught and eliminated in the development cycle, and the product is double tested again after the first bug elimination. This is not possible for the Waterfall method, since the product is tested only at the very end, which means any bugs found results in the entire program having to be re-written.

3. Agile's modular nature means employing better suited object-oriented designs and programs, which means one always has a working model for timely release even when it does not always entirely match customer specifications. Whereas, there is only one main release in the waterfall method and any problems or delays mean highly dissatisfied customers.

4. Agile methods allow for specification changes as per end-user's requirements, spelling customer satisfaction. As already mentioned, this is not possible when the waterfall method is employed, since any changes to be made means the project has to be started all over again.

5. However, both methods do allow for a sort of departmentalization e.g. in waterfall departmentalization is done at each stage. As for Agile, each coding module can be delegated to separate groups. This allows for several parts of the project to be done at the same time, though departmentalization is more effectively used in agile methodologies.

In conclusion, though on the plus side, waterfall's defined stages allow for thorough planning, especially for logical design, implementation and deployment, agile methodology is a sound choice for development of the product having volatile requirement and web design projects. More and more firms are becoming Agile.

Table 2:- Table showing difference between traditional and agile methodology

	Traditional View	**Agile Perspective**
Design Process	Deliberate and formal, linear sequence of steps, separate formulation and implementation, rule driven	Emergent, iterative and exploratory, knowing and action inseparable, beyond formal rules
Goal	Optimization	Adaptation, flexibility, responsiveness
Problem Solving Process	Selection of the best means to accomplish a given end through well-planned, formalized activities	Learning through experimentation and introspection, constantly reframing the problem and its solution
View of the Environment	Stable, predictable	Turbulent, difficult to predict, Adaptable

Key characteristics	• Control and direction • Avoids conflict • Formalizes innovation • Manager is controller • Design precedes implementation	• Collaboration and communication; integrates different worldviews • Encourages exploration and creativity; opportunistic • Manager is facilitator • Design and implementation are inseparable and evolve iteratively

Limitations of Agile Methods

Some important points highlighted by Nielsen[7] and Dias[11] about agile development are:

- The communication complexity grows proportionally to the size of the development team;
- Minimum documentation makes difficult the reuse of a particular artifact which, in addition, is not developed as a generic and reusable code;
- Assumption that customers are always available to: schedule meetings, participating, solving questions and making decisions together with the development team, is not always feasible in practice;
- Many issues arise during the implementation of interaction detailed design. Developers may not solve these issues in order to save time during development.
- Product development is divided into smaller steps that are accomplished one at a time. Users do not have the experience of integrating different features and therefore it may be impossible for them to work with this lack of integration.

2.4 Selection of a Life Cycle Model based on some important project characteristics

Software development lifecycle selection depends on multiple factors of the project. In this section we have discussed the lifecycle selection based on following factors[15] :-

- Requirements.
- Development team.

- Users.
- Project type and associated risk.

Based on characteristics of requirements:-

Selection of software development lifecycle is highly dependent on the characteristics of the requirements. For some project, requirements are volatile or difficult to understand or initially not complete. So, on the basis of these characteristics of requirement we have tabularized the suitability of lifecycle.

Table 3: Selection of a model based on characteristics of the requirements

Requirements	Waterfall	Prototype	Iterative Enhancement	Evolutionary development	Spiral	RAD	Agile
Are requirements easily understandable and defined?	Yes	No	No	No	No	Yes	No
Do we change requirements quite often?	No	Yes	No	No	Yes	No	Yes
Can we define requirements early in the cycle?	Yes	No	Yes	Yes	No	Yes	Yes
Requirements are indicating a complex system to be built	No	Yes	Yes	Yes	Yes	No	No

Based on status of development team:-

Lifecycle selection does not fully dependent on project characteristics but it depends on the characteristics of the development team also. Some of the team members have less experience, some of them having experienced but little domain knowledge, some of them having no experience on tools being used in the project. The table below shows the lifecycle suitability based on these characteristics.

Table 4: Selection of a model based on status of development team

Development Team	Waterfall	Prototype	Iterative Enhancement	Evolutionary development	Spiral	RAD	Agile
Less Experience on similar projects	No	Yes	No	No	Yes	No	No
Less domain knowledge (new to technology)	Yes	No	Yes	Yes	Yes	No	No
Less experience on tools to be used	Yes	No	No	No	Yes	No	No
Availability of training, if required	No	No	Yes	Yes	No	Yes	No

Based on user's participation:-

Selection of software development lifecycle also depends on user's involvement during the software development. The table below shows the lifecycle suitability based on the user's involvement during different lifecycle phases.

Table 5: Selection of lifecycle based on user's participation

Involvement of users	Waterfall	Prototype	Iterative Enhancement	Evolutionary development	Spiral	RAD	Agile
User Involvement in all phases	No	Yes	No	No	No	Yes	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No	No
Users are experts of problem domain	No	Yes	Yes	Yes	No	Yes	Yes

Based on type of project with associated risk:-

Lifecycle selection is highly dependent on the type of the project and risk associated with the project. For some project funding is stable, for some project we have to follow deadline strictly, and some project should be highly reliable. Based on these characteristics, we have tabularized the lifecycle suitability for a project.

Table 6: Lifecycle selection based on type of project with associated risk

Project type and risk	Waterfall	Prototype	Iterative Enhancement	Evolutionary development	Spiral	RAD	Agile
Project is the enhancement of the existing system	No	No	Yes	Yes	No	Yes	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes	Yes
Are resources (time, money, people) scarce?	No	Yes	No	No	Yes	No	No

2.5 Neural Networks

A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. Neural network learning is also referred to as connectionist learning due to the connections between units.

Neural networks involve long training times and are therefore more suitable for applications where this is feasible. They require a number of parameters that are typically best determined empirically, such as the network topology or “structure.” Neural networks have been

criticized for their poor interpretability. For example, it is difficult for humans to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network.

Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained. They can be used when you may have little knowledge of the relationships between attributes and classes. Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process.

Adaptation of neural network for our problem

An artificial neural network is composed of computational processing elements with weighted connections. We used feed forward multilayer perceptron network and the back propagation training algorithm (hence referred as Back Propagation Neural Network – BPN). The neural network architecture is designed using Matlab neural network tool box. The input layer has one neuron for each of the input variable (domain measures – Dj, j = 1,. . .,22). We used one hidden layer. There is one neuron in the output layer. The network learns by finding a vector of connection weights and minimizes the sum of squared errors on the training data set. One pass through all of the training observations (Training Phase) is called an epoch. The network is trained with a continuous back propagation learning algorithm; the weights are adjusted after each observation is fed forward. Various neural network architectures are tested. The number of units in the hidden layer, learning rate and momentum rate are adjusted to find a preferred combination. The activation function used is 'tansig'. Activation function is the logistic function, with a gain parameter that controls how sharply the function changes from zero to one. We trained the network, where the weights are adjusted after each epoch. Various values for the number of neurons are tested to find the final value. After training, the network is simulated for the validation data set (Testing Phase) and the classification outputs are obtained.

What is back propagation?

Backpropagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known target value. The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for prediction). For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name backpropagation). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops. The backpropagation algorithm performs learning on a multilayer feed-forward neural network. It iteratively learns a set of weights for prediction of the class label of tuples.

What is feed-forward neural network?

A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer. An example of a multilayer feed-forward network is shown in Figure. Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the input layer. These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of "neuronlike" units, known as a hidden layer. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers is arbitrary, although in practice, usually only one is used.

The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction for given tuples. The units in the input layer are called input units. The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units.

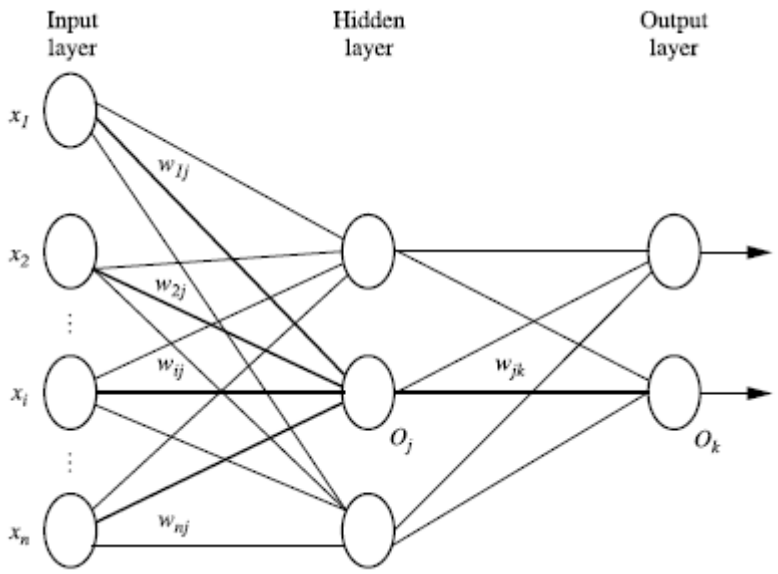


Fig.7: Layered architecture of neural network

we say that it is a two-layer neural network. (The input layer is not counted because it serves only to pass the input values to the next layer.) Similarly, a network containing two hidden layers is called a three-layer neural network, and so on. The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer. It is fully connected in that each unit provides input to each unit in the next forward layer. Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer. It applies a nonlinear (activation) function to the weighted input. Multilayer feed-forward networks, given enough hidden units and enough training samples, can closely approximate any function.

Chapter 3

Research Work

In this chapter, we propose a methodology for the selection of suitable lifecyle for a software project. Here we have identified the various important project and process metrics. We have assigned some weight to all of these metrics based on their impact on software development lifecycle selection. We have considered some sample project and identified the value of input for all of these metrics and then identified best suitable software development lifecycle for the project.

3.1 Framework for identification of factors leading to selection of software lifecycle model:-

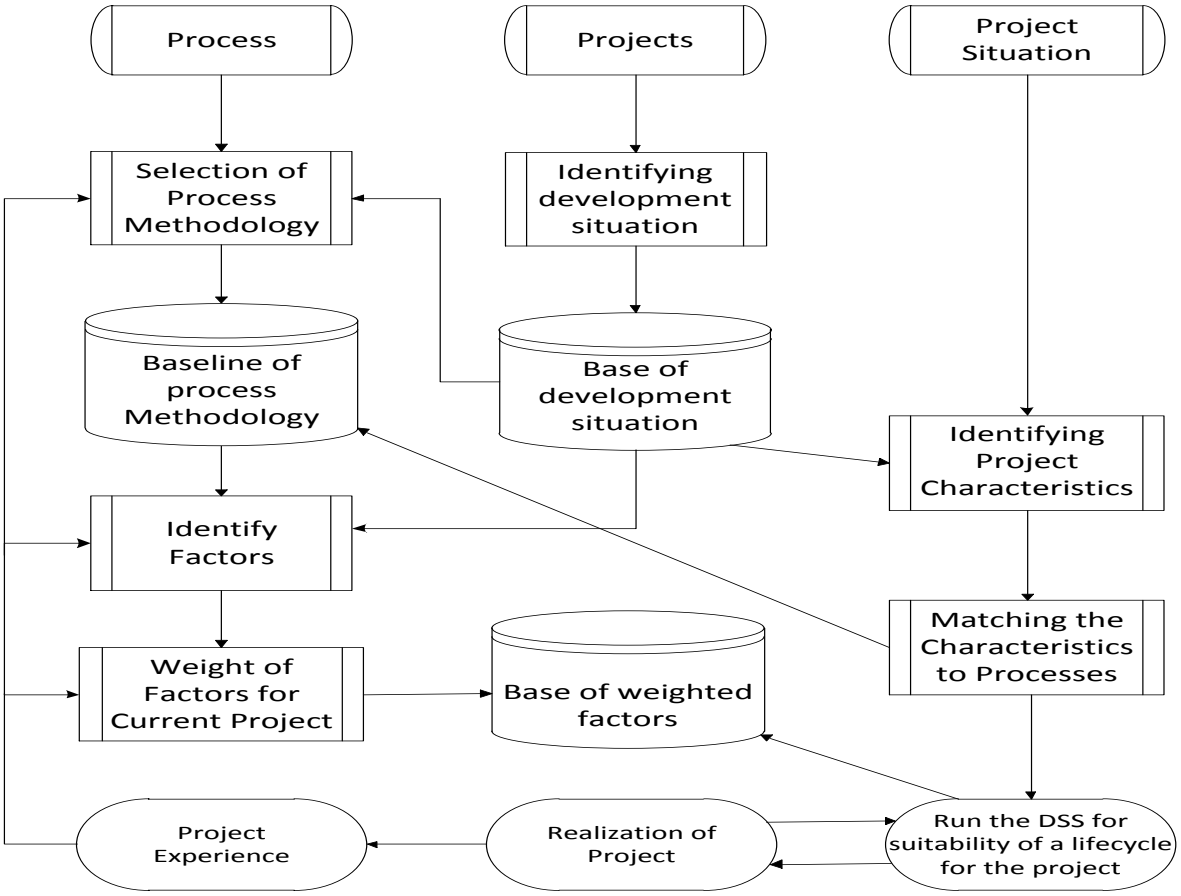


Fig. 8:- Framework for selection of software development lifecycle model

Description of the lifecycle selection framework:-

This picture is complete framework which depicts the sequence of operations of the project. There are three parts process, projects and project situation. There are three data storage classes that are used for storage of useful project characteristics and results, for future reference.

i. Base of development situation: - For large number of already developed projects, we will identify development situation and store it in the "Base of development situation" database.

ii. Baseline of process methodology: - It stores the process methodology to be followed for the current process, that is obtained by consulting base of development situation and the process under consideration.

iii. Base of weighted factors:- It stores the weighted factors of the project and provides these factors as input to the decision support system at runtime.

For a new project to be developed, we identify the project situation then identify project characteristics by consulting base of development situation. Now we will match these characteristics to processes by consulting baseline of process methodology. Now run the decision support system (DSS) for the current project and find the result.

At this stage we will have realization of project and project experience, which will be preserved for the future reference.

3.2 Description of weight distribution to different metrics:-

Decision support system is based on the weight and input values of a large number of identified metrics. Input parameters are divided in five categories. Table below describes the numerical equivalent value for different categories.

Table 7:- Numeric values for different input category

Category	Very Low	Low	Medium	High	Very High
Value	1	2	3	5	8

We have assigned weight to all the identified metrics with respect to the selection of an appropriate development lifecycle for the project. The numerical value assigned to weight is normalised with values ranging from 0.0 to 1.0. Weights are assigned in such manner so that the metrics having more support for traditional has been given more weight and the metrics having more support for agile has been given less weight. For different projects, different metric are important. Therefore, we have chosen the weights of the metrics depending upon the role of these metrics on the lifecycle development.

3.2.1 Weight distribution parameter:-

Weights are assigned in such manner so that the metrics having more support for traditional has been given more weight and the metrics having more support for agile has been given less weight.

1. Volatility of requirements:-

This software metric signifies the frequency of changing of requirement. For a given project if requirement changes very frequently then the value of this metric will be high and value will be very low if requirement will be stable.

The more value of this metric strongly supports agile development, so we have chosen very less (0.02) value as weight for this metric.

2. Complexity:-The project having large size and which require more effort is considered to be complex project. That is, the development process for those projects will be very complex and it requires a detailed analysis and a systematic way of development.

More value of this metric strongly supports traditional methodology to follow, so we have assigned very high (0.1) value as weight for this metric.

3. Business Risk:-

This metric is related to return on investment and customer satisfaction. For example, suppose customer is unsatisfied with the product after release and hence it has no market value then organization should be able to release new version but it will be very time consuming and costly for organization. In this case risk is high and organization will suffer heavy loss.

More value of business risk supports for agile development because in agile development customer is always available during development and product is released in increments not in the end, so any deficiency can be detected early.

4. Technical Risk:-

Technical risk involves the non-availability of developer, non-availability of technology that is tools etc. during development. It may occur due to failure of tool during development or leaving of developer before completion of task.

5. Operational Risk:-

This is the risk involved due to failure of some functionality of the project. If the impact of such failure is very high then we will say that operational risk is high.

For example, suppose in some safety system if any functionality fails then their impact will be very high so, operational risk is high.

More value of this metric supports traditional methodology because these types of systems should be designed is a systematic and properly defined way. So, we have assigned very high (0.1) value as weight for this metric.

6. Flexibility:-

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

Agile methodology is best suited for the project having more flexibility because it will be easy to develop and deliver software in increments.

So, we have chosen less weight for this metric.

7. Modularization of Task:-

Modularization is very important for quick and easy software development. If tasks are divided in modules then it will be very easy to develop the modules in parallel for quick release. Agile methodology is more suitable for development, if tasks can be divided in modules. So, less weight assigned to this metric.

8. Time to Market:-

This metric signifies the time (in months) before which at least first phase (least functionality) of the product must be released. If the value of this metric is less for a project, then agile methodology is suitable for the development. So, the weight assigned is less.

9. Amount of requirement known initially:-

It is not possible to know all the requirements initially for several projects. Some requirements are visible only after using the minimum workable (first release) of software.

If less number of requirements is known initially then we should follow agile methodology because customer is always involved and they can add requirements at later stages when they realize the requirement. So, less weight is given to this metric.

10. Clarity and Completeness of requirement:-

If the requirement is very well defined, clearly visible and does not require any further analysis then we will say that it is clear and complete.

11. Expandability: - The ease with which changes can be made to the software at later stages. The more value of this metric has more support for agile methodology.

12. Coupling: - The degree of interdependence between classes. Coupling increases complexity and hence more value of this metric has more support for traditional methodology.

13. Tool Experience: - How much year of work experience the developer has on the tool to be used?

14. Platform volatility: - How frequently the platform (Operating system for which product is being developed) is changing. For example, if we are developing windows based project then it require frequent modification because Microsoft releases new version of windows frequently.

15. Application Experience: - what is the work experience of the developer on the desired application (application may be java or c etc.).

16. Programmer's capability: - How much capable the programmer is; for development of the project?

17. Add-on Function:- How much percent of functions to be developed are Add-on functions. Add-on functions are fancy functions.

18. Necessary Functions:- These are essential functions which should be developed in a defined manner.

19. Reuse of existing code:- In the development of current project, the amount of code taken from existing code.

20. Develop for reuse:- If a project is to be developed as a base project then it should be developed in a defined way and should be well documented. Quality of such product should be very high.

21. Platform experience:- How much work experience developers have on the platform to be used for current project

22. Tool experience:- How much work experience developers have on the tools being used for current project? Tool may be Net beans, eclipse, meta-edit, rational rose.

3.3 Criteria for selection of input category for a metrics for project under consideration out of five possible categories:-

We cannot measure all the metrics on the same scale. There should be different measurement scale for different metrics on the basis of their characteristics. For example, we cannot measure length, area and volume on the same scale.

Measurement parameter for identification of values of different metrics:-

1. Volatility of Requirement:-

Table 8:- Table showing criteria for selection of values for "volatility of requirement"

How much percent of known requirements are volatile	Category
Less than 10 %	Very Low
10-19 %	Low
20-29 %	Medium
30-39 %	High
Greater than 39 %	Very High

2. Complexity:-

This metric can be measured by object-oriented metrics given by chidamber and kemerer[4].

There are three metrics which is useful for measurement of Complexity. These are WMC, NOC and DIT.

WMC:- Weighted Methods Per Class(Method Count for a class)

WMC = Number of methods defined in a class.

It is difficult to reuse classes with many methods. WMC is useful to predict the time and effort required to develop and maintain the class.

Table 9:- Values for complexity based on WMC

How much percent of class has more than 24 methods	Category
Less than 10 %	Very Low
10-14 %	Low

15-19 %	Medium
20-24 %	High
Greater than 24 %	Very High

DIT:- Depth of Inheritance Tree

DIT= Maximum inheritance path from the class to the root class.

The deeper a class is in hierarchy, the more methods & variables it is likely to inherit, making it more complex.

A recommended DIT is 5 or less. Excessively deep class hierarchies are complex to develop.

Table 10:- Values for complexity based on DIT

DIT	**Complexity**
1, 2	Very Low
2, 3	Low
4, 5	Medium
6, 7	High
Greater than 7	Very High

3. Business Risk:-

Risk that has direct impact on business value of the product comes under business risk such as unexpected changes in revenue, unexpected changes in costs from those budgeted, the unit sales that are less than forecast and unexpected development costs. Business risk is influenced by numerous factors, including sales volume, per-unit price, input costs,

competition, and overall economic climate and government regulations. So, the value of this metric should be chosen by considering all the above said factors.

4. Technical Risk:-

Technical risks are ranging from software glitches to power outages to viruses that can completely shut down a firm's operations. These are serious risks that a firm must plan to face. Risk involved with installing new system also comes under technical risk. When a firm switches over to a new system without proper integration, the new system is unable to perform all that was promised and sometimes even performs worse than the system it was replacing. New system often requires employees to operate according to new processes. These may be difficult to learn, take training to execute correctly, or may even be outright resisted by employees who prefer the old way of doing business. So, the value of this metric should be chosen by considering all the above said factors.

5. Operational Risk:-

Operational risk is the risk of loss resulting from inadequate or failed internal processes, people and systems, or from external events. Causes of operational risks include failure to address priority conflicts, failure to resolve the responsibilities, insufficient resources, no proper subject training, no resource planning and lack of communication in team. So, the value of this metric should be chosen by considering all the above said factors.

6. Add-on Function:-

Table 11:- Criteria for selection of values for “add-on function”

How much percent of functions to be developed are Add-on functions	Category
Less than 20 %	Very Low
20-39 %	Low
40-59 %	Medium
60-79 %	High
Greater than 79 %	Very High

7. Necessary Function:-

Table 12:- Criteria for selection of values for “necessary function”

How much percent of functions to be developed are Necessary functions	Category
Less than 20 %	Very Low
20-39 %	Low
40-59 %	Medium
60-79 %	High
Greater than 79 %	Very High

8. Flexibility:-

Modifying the source code is very easy, but it is very difficult to manage the impact of changes on the other parts of the source code. Flexibility is the effort required to modify an operational program.

Table 13:- Table showing criteria for selection of values for “flexibility”

Percentage change required in source code due to addition of new functions (on an average)	**Category**
Less than 5 %	Very High
5-9 %	High
10-14 %	Medium
15-19 %	Low
Greater than 20 %	Very Low

9. Modularization of Task:-

This metric can be measured by object-oriented metrics given by chidamber and kemerer.

The CBO(Coupling Between Object) is useful for measurement of this metric.

CBO=Number of classes to which a class is coupled.

Two classes are coupled when methods declared in one class uses methods or instance variables defined by the other class. Modularization of task is not possible if coupling between object classes are excessive. CBO greater than 14 is too high.

Table 14:- Criteria for selection of values for “modularization of task”

Value of CBO	Modularization of Task
Less than 2	Very High
2, 3	High
4, 5	Medium
6,7	Low
Greater than 7	Very Low

10. Time to Market:-

Table 15:- Table showing criteria for selection of values for “time to market"

Time (in month) before which minimum workable product must be released	Time to Market
2 Month	Very High
4 Month	High
6 Month	Medium
8 Month	Low
Greater than 8 Months	Very Low

11. Amount of requirement known initially:-

Table 16:- Criteria for selection of values for “amount of requirement known initially”

Amount of requirement known initially (Percentage)	Category
Less than 20 %	Very Low
20-39 %	Low
40-59 %	Medium
60-79 %	High
Greater than 79 %	Very High

12. Clarity and Completeness of requirements:-

If the requirement is very well defined, clearly visible and does not require any further analysis then we will say that it is clear and complete.

Table 17:- Criteria for selection of values for "clarity and completeness of requirements"

How much amount of known requirements are clear and complete (Percentage)	**Category**
Less than 20 %	Very Low
20-39 %	Low
40-59 %	Medium
60-79 %	High
Greater than 79 %	Very High

13. **Expandability:-** The effort required in addition of new functionality to the already working software. The value of this metric can be selected based on the effort estimation for addition of new functionality

14. Coupling:- The CBO(Coupling Between Object) is useful for measurement of this metric.

CBO=Number of classes to which a class is coupled.

Two classes are coupled when methods declared in one class uses methods or instance variables defined by the other class. The value of CBO greater than 14 is too high. So, coupling will be very high for CBO greater than 14.

Table 18:- Table showing criteria for selection of values for “coupling”

Value of CBO	Coupling
Less than 2	Very Low
2, 3	Low
4, 5	Medium
6,7	High
Greater than 7	Very High

15. **Programmer Capability:-**We cannot define any specific range for this metric. It will depend on the capability of the development team and can be categorized on the basis of current situation.

16. **Application Experience:-**How much experience developers have on the desired application? Application may be different programming language like c, c++, java.

Table 19:- criteria for selection of values for “application experience”

Time (in month)	Application Experience
Less than 12 Month	Very Low
12 Month – 24 Month	Low
24 Month- 30 Month	Medium
30 Month- 36 Month	High
Greater than 36 Months	Very High

17. **Reuse of existing code:-** In the development of current project, the amount of code taken from existing code.

Table 20:- criteria for selection of values for "reuse of existing code"

How much amount of code taken from existing code (Percentage)	**Category**
Less than 20 %	Very Low
20-39 %	Low
40-59 %	Medium
60-79 %	High
Greater than 79 %	Very High

18. **Develop for reuse:-** Is this project is developed as a base project? If a project is to be developed as a base project then it should be developed in a defined way and should be well documented. Quality of such product should be very high.

Table 21:- criteria for selection of values for "develop for reuse"

Purpose of the project	**Develop for reuse**
Developed as a base project (that is, developed only for reuse)	Very High
Probability of being used in other project is very high	High
It may require addition of some functionality at later stages	Medium
At this point, there is no visible project which will require code of this project	Low
It can never be reused	Very Low

19. **Platform Volatility:-** How frequently the platform (Operating system for which product is being developed) is changing. For example, if we are developing windows based project then it require frequent modification because Microsoft releases new version of windows frequently.

Table 22:- criteria for selection of values for platform volatility

Type of changes in platform	**Platform volatility**
Likely to change from one platform to another having different architecture (windows to linux)	Very High
Likely to change from one platform to another having same architecture (windows xp to windows 7) or (red hat to Ubuntu)	High
Likely to change from one platform to another having different version only (windows xp service pack 2 to windows xp service pack 3) or (Ubuntu 10 to Ubuntu 11)	Medium
At this point, there is no visible change but it might change at later stages	Low
It will never change	Very Low

20. **Platform Experience:-** How much work experience developers have on the platform to be used for current project?

Table 23:- criteria for selection of values for platform experience

Developers work experience on the platform to be used for current project? Time (in month)	**Platform Experience**
Less than 6 Month	Very Low
6 Month – 12 Month	Low
12 Month- 18 Month	Medium
18 Month- 24 Month	High
Greater than 24 Months	Very High

21. Tool Experience:-

How much work experience developers have on the tools being used for current project? Tool may be Net beans, eclipse, meta-edit, rational rose.

Table24:- criteria for selection of values for tool experience

Developers work experience on the tool to be used for current project? Time (in month)	**Tool Experience**
Less than 6 Month	Very Low
6 Month – 12 Month	Low
12 Month- 18 Month	Medium
18 Month- 24 Month	High
Greater than 24 Months	Very High

22. **Team Cohesion:-** Ease of communication and interaction among team members is known as team cohesion. The value for this metric can be chosen on the basis of current situation of the organization and team members.

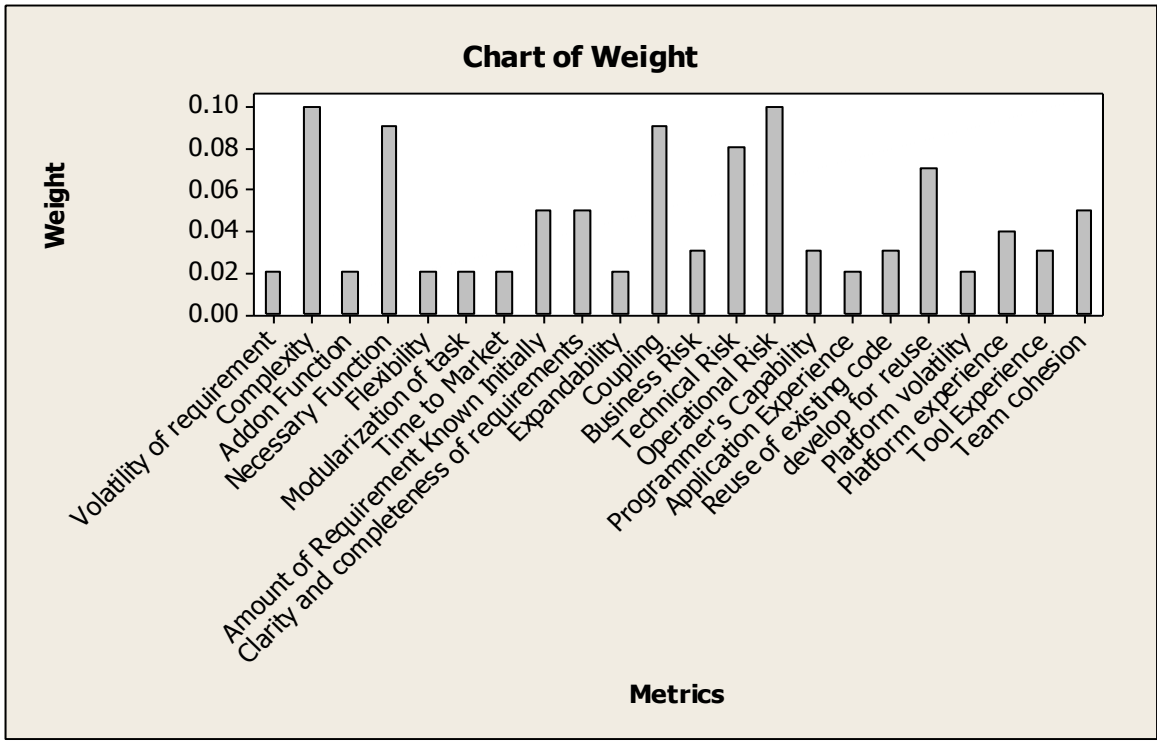


Fig.9:- Graph showing weight distribution to different metrics

3.4 Proposed Algorithm :-

Step 1:- Assign input values to each metric for a given project from the five possible values.

Category	Very Low	Low	Medium	High	Very High
Value	1	2	3	5	8

Step 2:- Calculate S = $\sum_{i=1}^{22} w_i * p_i$.

Where w_i is the weight assigned to the i^{th} metrics which is fixed (constant).

And p_i is the input values chosen for i^{th} metrics which is variable (project specific).

Step 3:- Select best suitable methodology for the given project on the basis of the value of S in step2.

If the value is between1 to 4 then we will follow agile software development lifecycle. If the value is 5 to 8 then we will follow traditional methodology. If value comes between 4 and 5 then we can choose any methodology or hybrid methodology.

3.5 Demonstration of algorithm using example projects

We have considered four example projects to demonstrate the working of decision support system.

1. Mobile App. Development (MAD).
2. Air Traffic Controller (ATC).
3. ERP implementation for small and medium enterprises (SME's).
4. Banking application development.

1. **Mobile Application Development:-** Input values are assigned to each of the metric and then the product of each input with their respective weight is calculated. Further, the sum of all these products is calculated and this sum will be the parameter for decision making. Now our output will range from 1 to 8. If the value is between1 to 4 then we will follow Agile Software development lifecycle. If the value is 5 to 8 then we will follow traditional methodology.

Table 25:- Weight distribution, input values, their product and total sum for mobile application development

S. no.	Metrics	Weight	Input Values (Mobile App. Development)	Product of weight & Input values for Mobile App. Development
1.	Volatility of requirement	0.02	Medium(3)	0.06
2.	Complexity	0.1	Very Low(1)	0.1
3.	Add-on Function	0.02	Medium(3)	0.06
4.	Necessary Function	0.09	Medium(3)	0.27
5.	Flexibility	0.02	High(5)	0.1
6.	Modularization of task	0.02	High(5)	0.1
7.	Time to Market	0.02	High(5)	0.1

8.	Amount of Requirement Known Initially	0.05	Medium(3)	0.15
9.	Clarity and completeness of requirements	0.05	High(5)	0.25
10.	Expandability	0.02	High(5)	0.1
11.	Coupling	0.09	Very Low(1)	0.09
12.	Business Risk	0.03	Very High(8)	0.24
13.	Technical Risk	0.08	Very Low(1)	0.08
14.	Operational Risk	0.1	Low(2)	0.2
15.	Programmer's Capability	0.03	Medium(3)	0.09
16.	Application Experience	0.02	Medium(3)	0.06
17.	Reuse of existing code	0.03	Medium(3)	0.09
18.	develop for reuse	0.07	Medium(3)	0.21
19.	Platform volatility	0.02	Low(2)	0.04
20.	Platform experience	0.04	Low(2)	0.08
21.	Tool Experience	0.03	Medium(3)	0.09
22.	Team cohesion	0.05	Medium(3)	0.15
	Total(Sum of product)			**2.71**

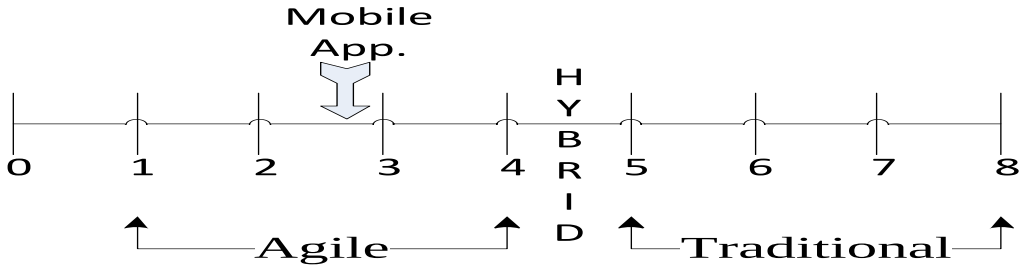


Fig.10:- Scale showing the output of mobile application development

Here for mobile application development the output is 2.71, so it indicates that agile methodology is best suited for their development.

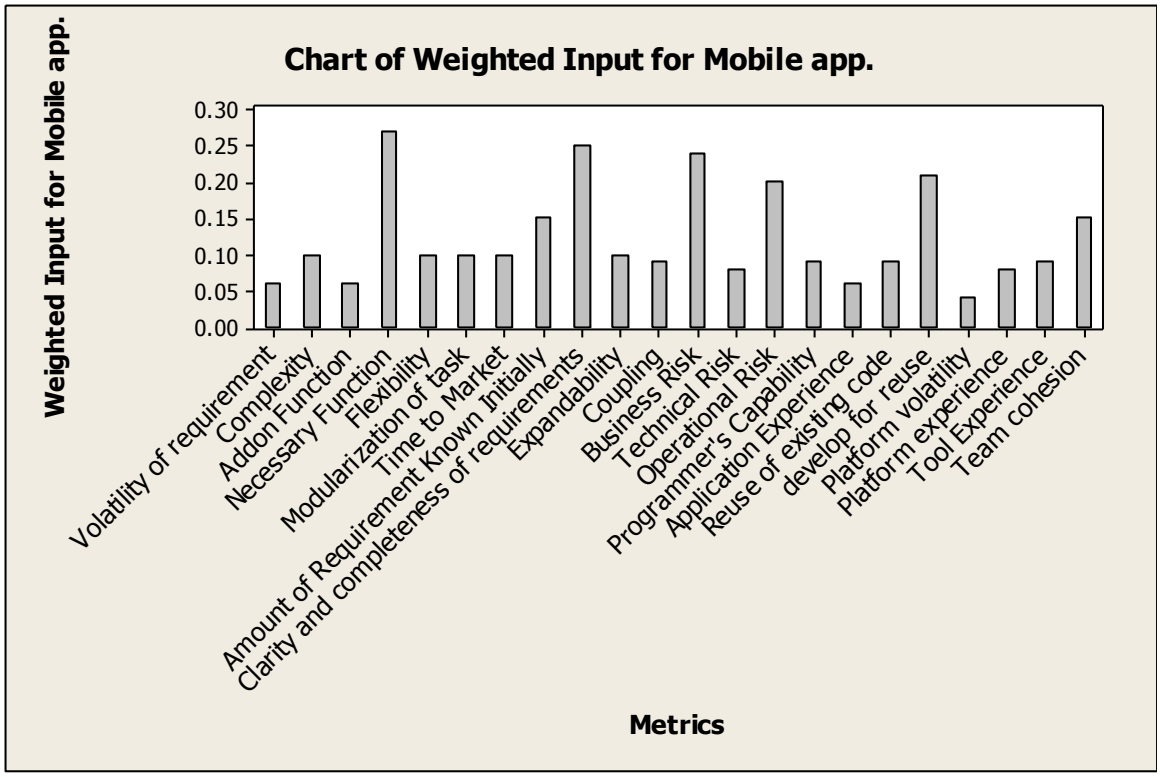


Fig.11:- Graph showing weighted input for mobile app. development

2. Air Traffic Controller (ATC)

Table 26:- Weight distribution, input values, their product and total sum for ATC

S. no.	Metrics	Weight	Input Values (Air Traffic Controller)	Product of weight & Input values for ATC
1.	Volatility of requirement	0.02	Very Low(1)	0.02
2.	Complexity	0.1	Very High(8)	0.8
3.	Add-on Function	0.02	Low(2)	0.04
4.	Necessary Function	0.09	Very High(8)	0.72
5.	Flexibility	0.02	Low(2)	0.04
6.	Modularization of task	0.02	Very Low(1)	0.02
7.	Time to Market	0.02	Low(2)	0.04
8.	Amount of Requirement Known Initially	0.05	Very High(8)	0.4
9.	Clarity and completeness of requirements	0.05	Very High(8)	0.4
10.	Expandability	0.02	Low(2)	0.04
11.	Coupling	0.09	High(5)	0.45
12.	Business Risk	0.03	Low(2)	0.06
13.	Technical Risk	0.08	Very High(8)	0.64
14.	Operational Risk	0.1	Very High(8)	0.8
15.	Programmer's Capability	0.03	Very High(8)	0.24
16.	Application Experience	0.02	Low(2)	0.04
17.	Reuse of existing code	0.03	Very Low(1)	0.03
18.	develop for reuse	0.07	High(5)	0.35
19.	Platform volatility	0.02	Low(2)	0.04
20.	Platform experience	0.04	Very High(8)	0.32
21.	Tool Experience	0.03	Very High(8)	0.24
22.	Team cohesion	0.05	Very High(8)	0.4
	Total(Sum of product)			**6.13**

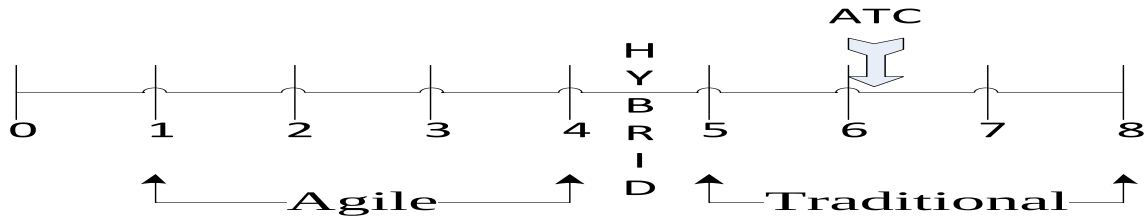


Fig.12:- Scale showing the output of ATC development

The output for air traffic controller is 6.13, which indicates that traditional methodology is best suited for their development.

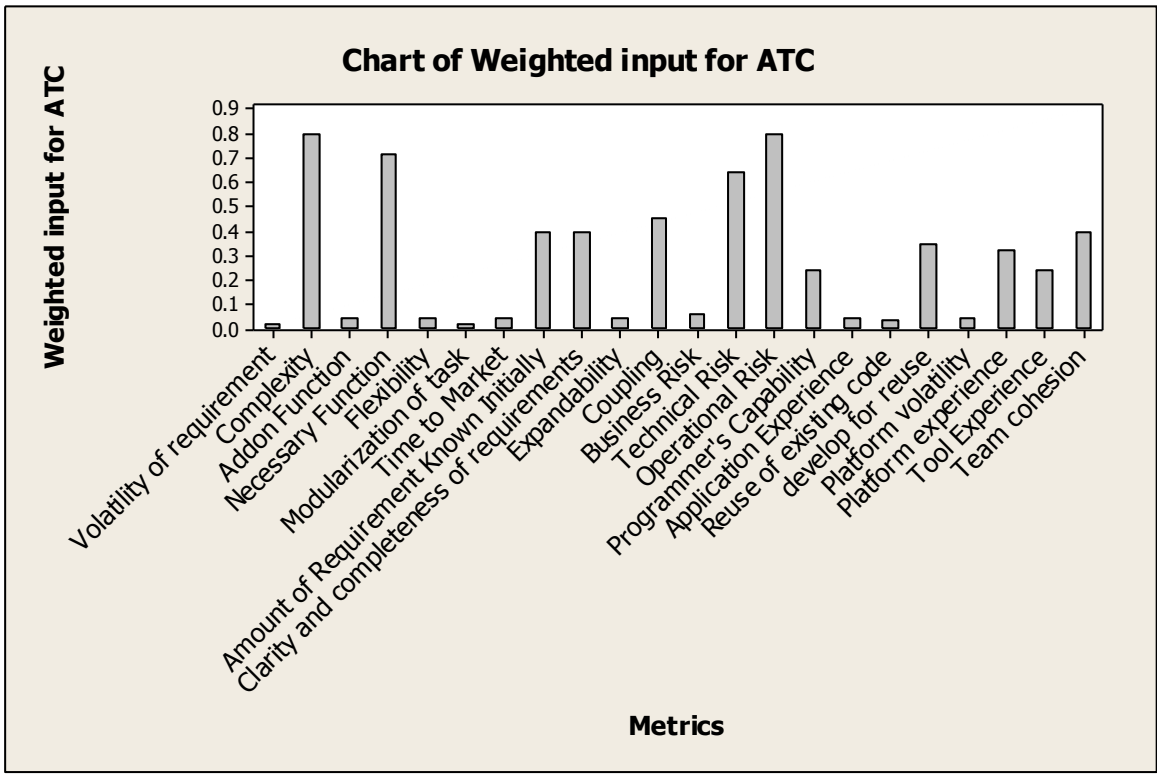


Fig. 13:- Graph showing weighted input for ATC

3. **ERP implementation for small and medium enterprises (SME's):-** Selection of lifecycle methodology for example projects ERP implementation in small and medium enterprises (SME's).

Table 27:- Weight distribution, input values, their product and total sum for ERP

S. no.	Metrics	Weight	Input Values (ERP)	Product of weight & Input values for ERP
1.	Volatility of requirement	0.02	Low(2)	0.04
2.	Complexity	0.1	High(5)	0.5
3.	Add-on Function	0.02	Low(2)	0.04
4.	Necessary Function	0.09	Very High(8)	0.72
5.	Flexibility	0.02	Medium(3)	0.06
6.	Modularization of task	0.02	High(5)	0.1

7.	Time to Market	0.02	High(5)	0.1
8.	Amount of Requirement Known Initially	0.05	High(5)	0.25
9.	Clarity and completeness of requirements	0.05	Medium(3)	0.15
10.	Expandability	0.02	Medium(3)	0.06
11.	Coupling	0.09	Low(2)	0.18
12.	Business Risk	0.03	Very High(8)	0.24
13.	Technical Risk	0.08	Medium(3)	0.24
14.	Operational Risk	0.1	High(5)	0.5
15.	Programmer's Capability	0.03	High(5)	0.15
16.	Application Experience	0.02	Low(2)	0.04
17.	Reuse of existing code	0.03	Medium(3)	0.09
18.	develop for reuse	0.07	Medium(3)	0.21
19.	Platform volatility	0.02	Low(2)	0.04
20.	Platform experience	0.04	Medium(3)	0.12
21.	Tool Experience	0.03	Medium(3)	0.09
22.	Team cohesion	0.05	High(5)	0.25
	Total(Sum of product)			**4.17**

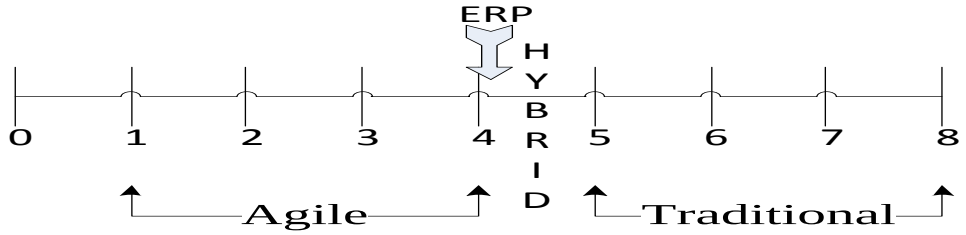


Fig.14:- Scale showing the output of ERP development

Here for ERP implementation in small and medium enterprises (SME's) the output is 4.17, so it indicates that hybrid methodology is best suitable for their development.

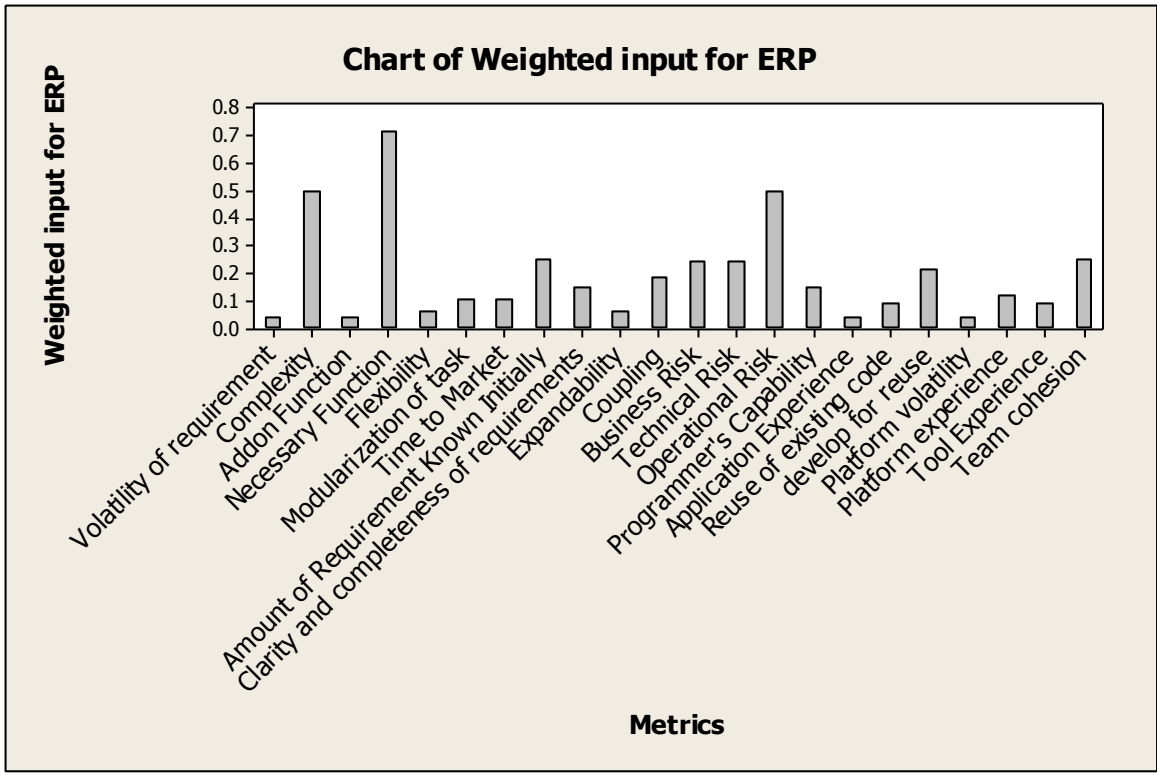


Fig. 15:- Graph showing weighted input for ERP

4. Banking application development

Table 28:- Weight distribution, input values, their product and total sum for banking application development

S. no.	Metrics	Weight	Input Values (Banking application)	Product of weight & Input values for Banking application
1.	Volatility of requirement	0.02	Very Low(1)	0.02
2.	Complexity	0.1	Medium(3)	0.3
3.	Add-on Function	0.02	Very Low(1)	0.02
4.	Necessary Function	0.09	Very High(8)	0.72
5.	Flexibility	0.02	Low(2)	0.04
6.	Modularization of task	0.02	Medium(3)	0.06
7.	Time to Market	0.02	High(5)	0.1
8.	Amount of Requirement Known Initially	0.05	High(5)	0.25
9.	Clarity and completeness of requirements	0.05	High(5)	0.25
10.	Expandability	0.02	High(5)	0.1
11.	Coupling	0.09	High(5)	0.45

12.	Business Risk	0.03	Very High(8)	0.24
13.	Technical Risk	0.08	High(5)	0.4
14.	Operational Risk	0.1	High(5)	0.5
15.	Programmer's Capability	0.03	Very High(8)	0.24
16.	Application Experience	0.02	Low(2)	0.04
17.	Reuse of existing code	0.03	Very Low(1)	0.03
18.	develop for reuse	0.07	Medium(3)	0.21
19.	Platform volatility	0.02	Very Low(1)	0.02
20.	Platform experience	0.04	Low(2)	0.08
21.	Tool Experience	0.03	Medium(3)	0.09
22.	Team cohesion	0.05	High(5)	0.25
	Total(Sum of product)			**4.41**

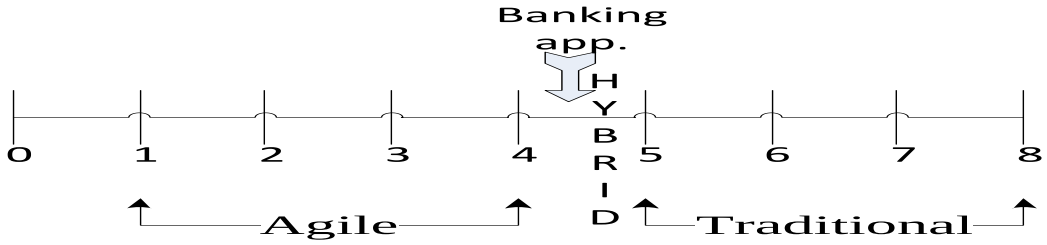


Fig.16:- Scale showing the output of banking application development

The output for banking application development is 4.41, so for this project hybrid methodology is best suitable methodology.

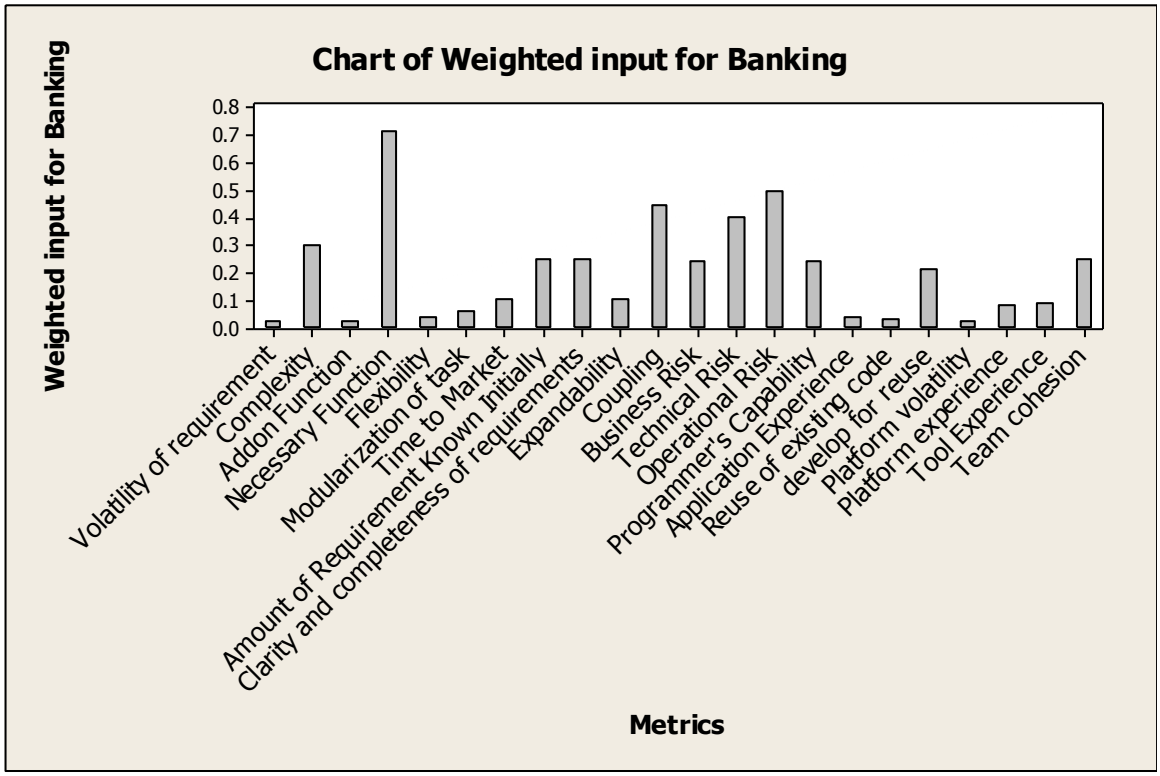


Fig. 17:- Graph showing weighted input for banking application development

Graph showing the comparison of weighted input among mobile app., ATC, ERP implementation and banking application:-

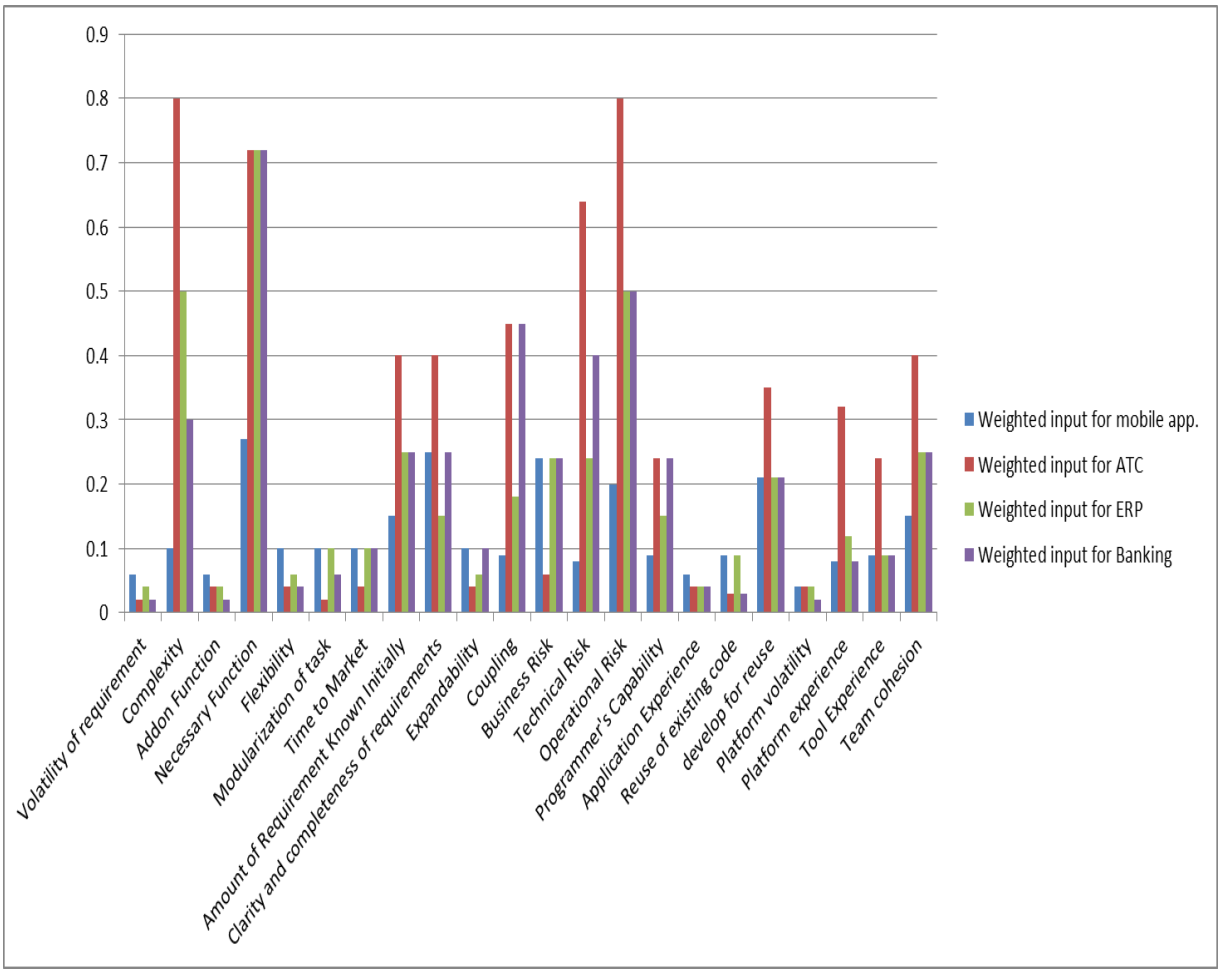
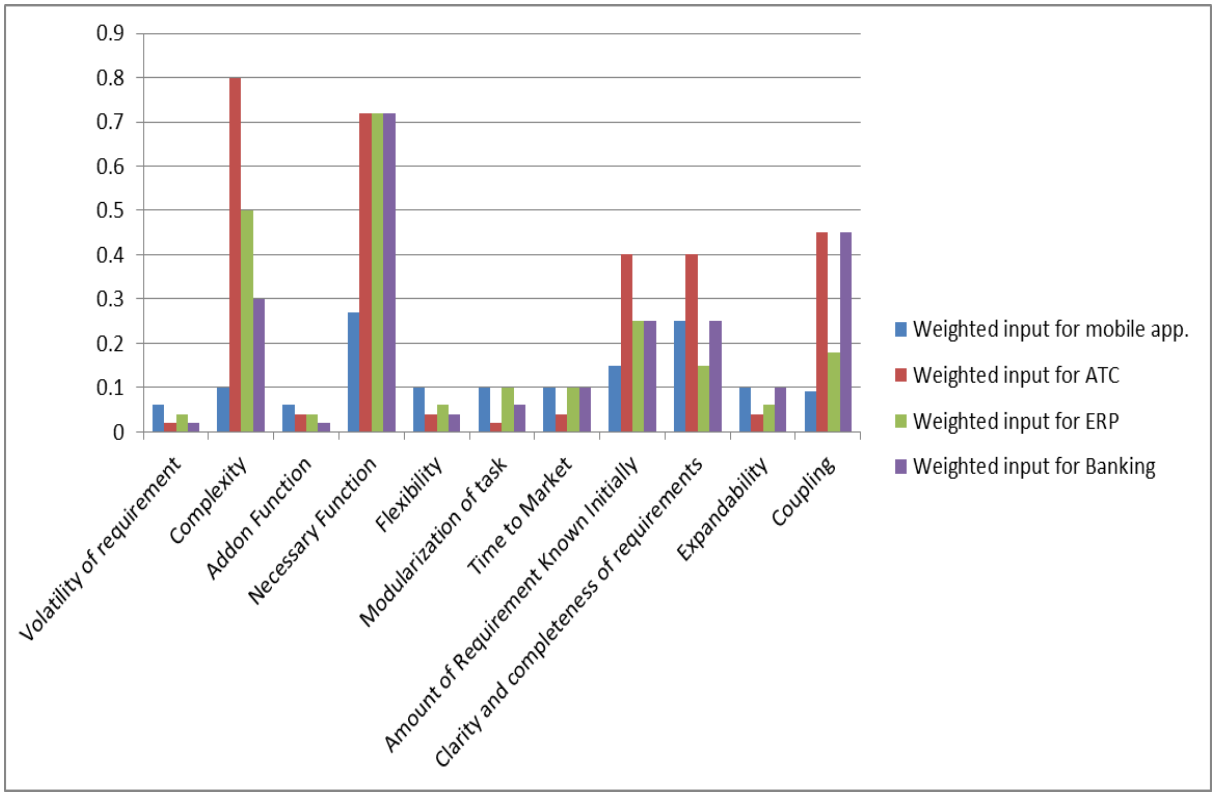


Fig. 18:- Comparison of weighted inputs between mobile app., ATC, ERP and banking application development



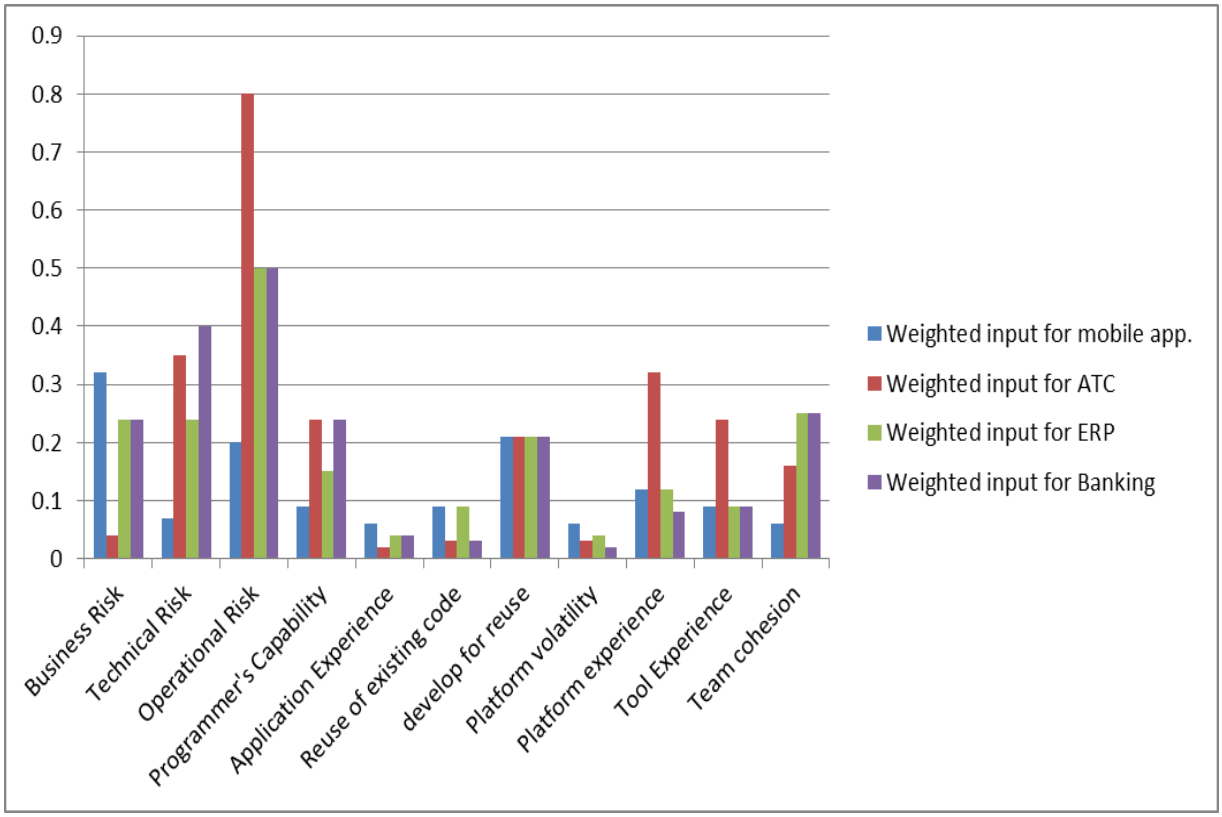


Fig. 19:- A more clear picture of comparison of weighted inputs between mobile app., ATC, ERP and banking application development

Chapter 4

A Neural Network approach for Estimating the Software Project Parameters and Identifying a Suitable Development Lifecycle

In this chapter, we have mapped the proposed method for lifecycle selection in chapter 3 with the neural network. We have used neural network tool available in Matlab to create, train and simulate the network.

Why use neural network

Neural network is used to simplify the complex problem like pattern recognition, trend analysis, and classification that are difficult tasks for humans. The beauty of neural network lies in its ability to learn from training samples and experience, self-organizing capability, real time operation, and its fault tolerance capability.

4.1 Problem mapped as neural network

The decision support system is simulated by three layer feed-forward back propagation neural network having input, hidden and output layer. Neural network tool available in Matlab is used for training and simulation. Network consists of twenty two neurons at input layer (number of inputs), three neurons at hidden layer and one neuron at output layer.

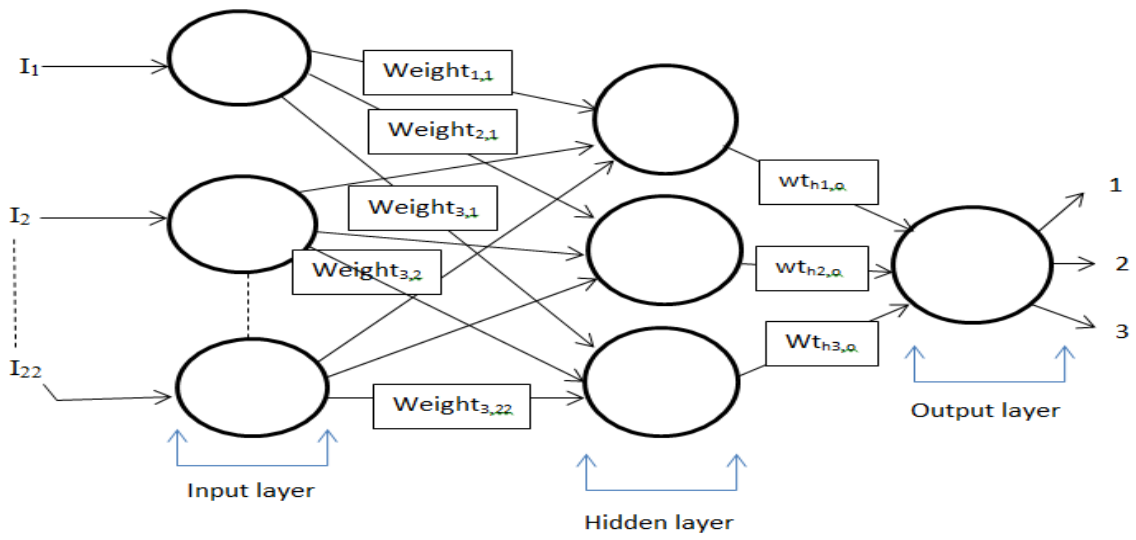


Fig. 20: Problem mapped as neural network

Output is divided into three categories 1, 2 and 3.Here we have mapped our previous output range as, the output range 1 to 4 is mapped as output 1, output range 4 to 5 is mapped as output 2 and output range 4 to 8 is mapped as output 3. If the output of network is 1 then it indicates that agile methodology is best suitable methodology for given input, output 2 indicates that both agile and traditional suites the given problem and output 3 indicates that traditional methodology is the best suitable methodology for the given problem.

4.2 Implementation of neural network

It has been implemented with the help of neural network tool available in neural network tool section in Matlab.

How to use neural network tool:-

To get started on a new problem, do the following:-

1. Import Input and Target data from the workspace with [IMPORT].
2. Create a new Network with [NEW].
3. Select the network in the Network list and click [OPEN]
4. Select the training tab in the Network Window.
5. Select training input and target data, and click [TRAIN].
6. You can also use the Network Window to simulate the network, or perform other tasks such as reinitialization and editing of weights.

Here are descriptions of each button:-

[IMPORT] - Imports data and networks from the workspace or a file.

[NEW] - Allows you to create a network or data.

[OPEN] - Opens the selected data or network for viewing and editing.

[EXPORT] - Exports data and networks to the workspace or a file.

[DELETE] - Removes the selected data or network.

The various steps involved in the training process is:-

1. **Create Network**:- For creation of new network, we have to choose network type, number of input neuron, number of hidden layer , number of output layer, input data, target data and transfer function. Our problem consists of 22 input therefore we have chosen 22 neurons at input layer, two neurons at hidden layer and one neuron at output layer.

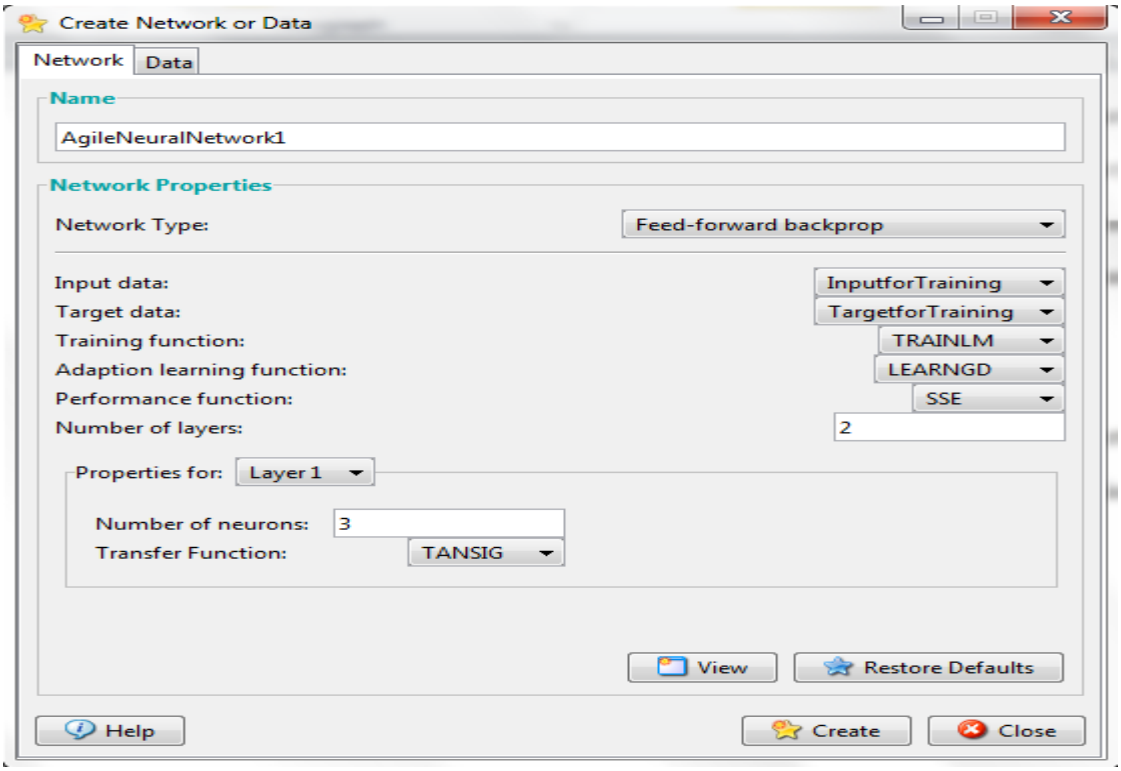


Fig. 21: Creation of network

Input and target data sample is generated from the four example projects discussed in chapter 3. Transfer function used is tangent sigmoid function. The equation for this function is tansig(n) = 2/(1+exp(-2*n))-1.

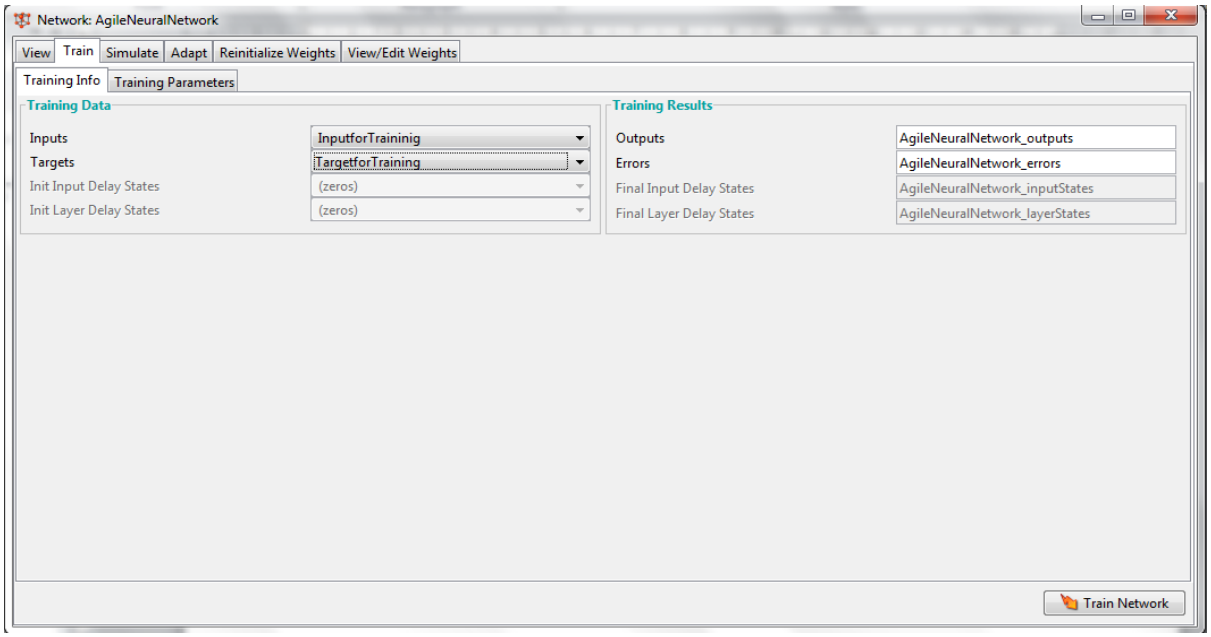


Fig. 22: Network Created

2. **Train Network**:- For training, we have to provide large number of input and their corresponding target, which we have generated from the four projects discussed in chapter 3 that is, mobile app. development, air traffic controller, ERP implementation in SMEs and banking system. The epoch chosen is 1000, it is the maximum number of iterations that the network has to perform during training.

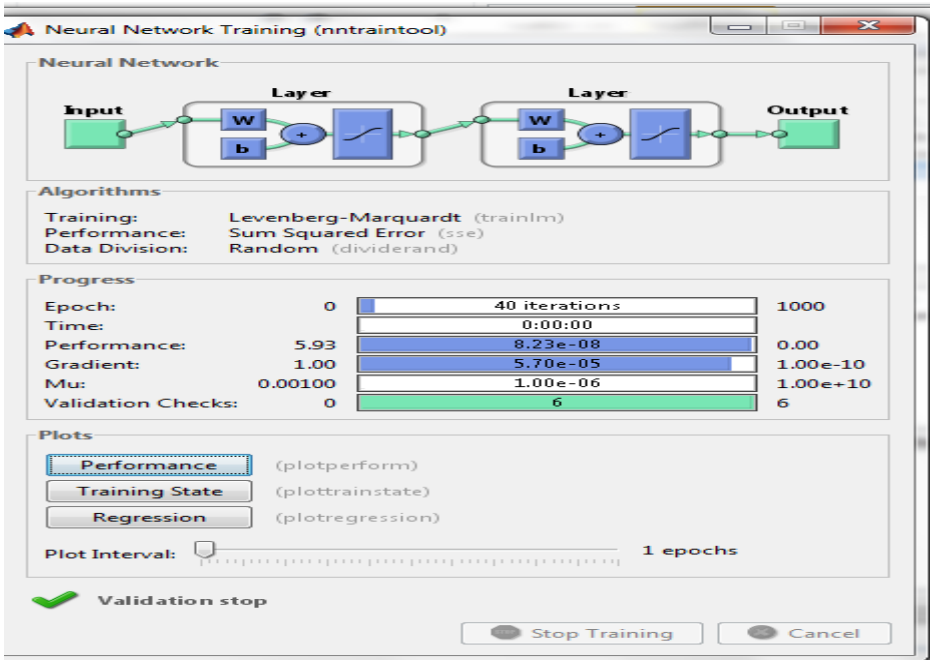


Fig. 23: Training of network

Final weight of the network after training:-

Table 29:-Final weight adjusted by neural network

S. no.	Metrics	Weight for input to hidden neuron 1	Weight for input to hidden neuron 2	Weight for input to hidden neuron 3
1.	Volatility of requirement	0.19376	0.89165	-0.40272
2.	Complexity	0.51028	-0.030383	-0.68257
3.	Add-on Function	-0.29233	-0.099261	0.067508
4.	Necessary Function	0.58891	-1.0546	0.41018
5.	Flexibility	0.30301	0.29065	0.3696
6.	Modularization of task	-0.48088	-0.042034	0.42978
7.	Time to Market	0.17812	0.41011	0.44175
8.	Amount of Requirement Known Initially	-0.3488	0.21002	0.30634
9.	Clarity and completeness of requirements	0.29833	0.4106	0.2556
10.	Expandability	-0.14357	0.27188	0.1635
11.	Coupling	0.33175	-0.90402	-0.070217
12.	Business Risk	-0.57239	-0.60949	0.597
13.	Technical Risk	0.31599	-0.38916	0.38757
14.	Operational Risk	-0.34957	-0.35567	-0.29739
15.	Programmer's Capability	0.3903	-0.11019	-0.20646
16.	Application Experience	0.059763	0.21941	0.23257
17.	Reuse of existing code	0.20259	0.4015	-0.44519
18.	develop for reuse	0.29662	0.31654	-0.60989
19.	Platform volatility	-0.27396	-0.035285	0.15748
20.	Platform experience	-0.045341	0.15639	-0.60084
21.	Tool Experience	0.36862	-0.1564	-0.29208
22.	Team cohesion	0.31846	0.26222	0.2154

Bias to hidden layer neuron:- [-1.3528; -0.40062; -1.2345]

Weight from hidden layer to output layer :- [0.69954 -1.4034 -0.88563]

Bias to output layer neuron :- [0.11863]

3. Simulation of network

Ones the network is trained , it can be simulated by the desired data. We can also check the correctness of network by giving input as those data for which output is known to us. If the network will give the output same as desired output then we can say that network is perfect.

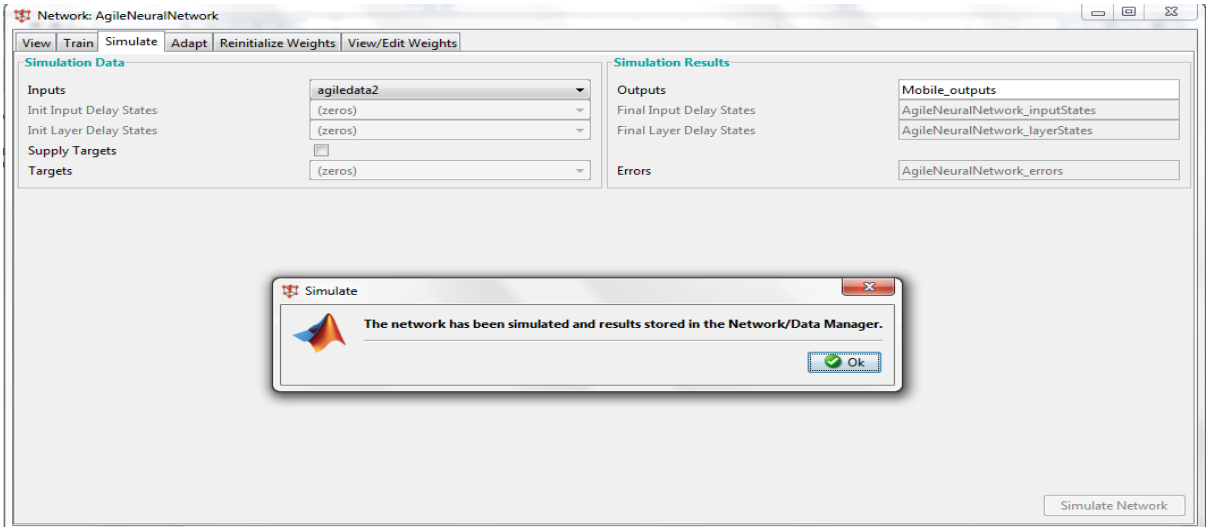


Fig. 24: Simulation of neural network

4.3 Case Study

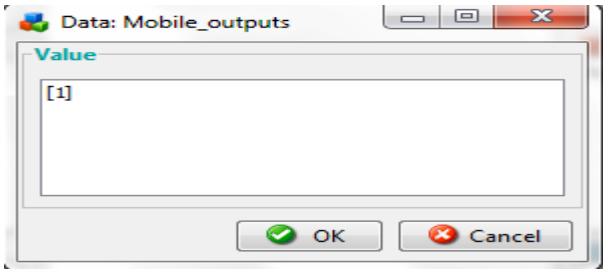
Case 1:- Input values for Mobile app development as data to the neural network tool

S. no.	Metrics	Input Values (Mobile App. Development)
1.	Volatility of requirement	Medium(3)
2.	Complexity	Very Low(1)
3.	Add-on Function	Medium(3)
4.	Necessary Function	Medium(3)
5.	Flexibility	High(5)
6.	Modularization of task	High(5)
7.	Time to Market	High(5)
8.	Amount of Requirement Known Initially	Medium(3)
9.	Clarity and completeness of requirements	High(5)
10.	Expandability	High(5)
11.	Coupling	Very Low(1)
12.	Business Risk	Very High(8)
13.	Technical Risk	Very Low(1)
14.	Operational Risk	Low(2)
15.	Programmer's Capability	Medium(3)
16.	Application Experience	Medium(3)
17.	Reuse of existing code	Medium(3)
18.	develop for reuse	Medium(3)
19.	Platform volatility	Low(2)
20.	Platform experience	Low(2)
21.	Tool Experience	Medium(3)
22.	Team cohesion	Medium(3)

Input:- 3;1;3;3;5;5;5;3;5;5;1;8;1;2;3;3;3;3;2;2;3;3

Target:-1(i.e Agile)

Output Screen:-



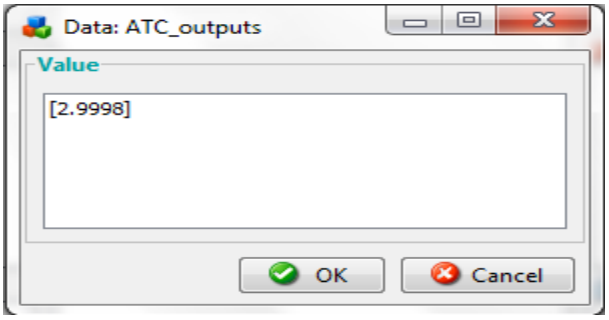
Case 2:- Input values for Air Traffic Controller as data to the neural network tool

S. no.	Metrics	Input Values (Air Traffic Controller)
1.	Volatility of requirement	Very Low(1)
2.	Complexity	Very High(8)
3.	Add-on Function	Low(2)
4.	Necessary Function	Very High(8)
5.	Flexibility	Low(2)
6.	Modularization of task	Very Low(1)
7.	Time to Market	Low(2)
8.	Amount of Requirement Known Initially	Very High(8)
9.	Clarity and completeness of requirements	Very High(8)
10.	Expandability	Low(2)
11.	Coupling	High(5)
12.	Business Risk	Low(2)
13.	Technical Risk	Very High(8)
14.	Operational Risk	Very High(8)
15.	Programmer's Capability	Very High(8)
16.	Application Experience	Low(2)
17.	Reuse of existing code	Very Low(1)
18.	develop for reuse	High(5)
19.	Platform volatility	Low(2)
20.	Platform experience	Very High(8)
21.	Tool Experience	Very High(8)
22.	Team cohesion	Very High(8)

Input:- 1;8;2;8;2;1;2;8;8;2;5;2;8;8;8;2;1;5;2;8;8;8

Target:-3(i.e Traditional)

Output Screen:-



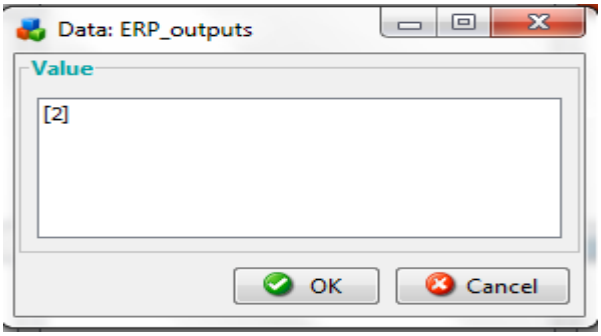
Case 3:- Input values for ERP implementation in SMEs as data to the neural network tool

S. no.	Metrics	Input Values (ERP)
1.	Volatility of requirement	Low(2)
2.	Complexity	High(5)
3.	Add-on Function	Low(2)
4.	Necessary Function	Very High(8)
5.	Flexibility	Medium(3)
6.	Modularization of task	High(5)
7.	Time to Market	High(5)
8.	Amount of Requirement Known Initially	High(5)
9.	Clarity and completeness of requirements	Medium(3)
10.	Expandability	Medium(3)
11.	Coupling	Low(2)
12.	Business Risk	Very High(8)
13.	Technical Risk	Medium(3)
14.	Operational Risk	High(5)
15.	Programmer's Capability	High(5)
16.	Application Experience	Low(2)
17.	Reuse of existing code	Medium(3)
18.	develop for reuse	Medium(3)
19.	Platform volatility	Low(2)
20.	Platform experience	Medium(3)
21.	Tool Experience	Medium(3)
22.	Team cohesion	High(5)

Input:- 2;5;2;8;3;5;5;5;3;3;2;8;3;5;5;2;3;3;2;3;3;5

Target:-2(i.e Both of them are suitable)

Output Screen:-



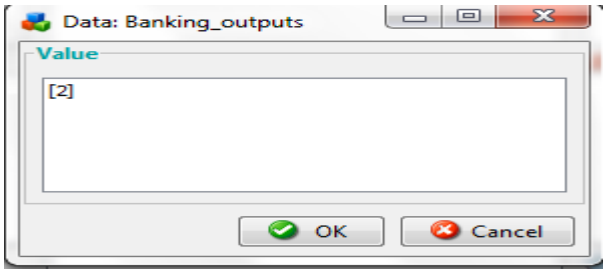
Case 4:- Input values for Banking Application development as data to the neural network tool

S. no.	Metrics	Input Values (Banking application)
1.	Volatility of requirement	Very Low(1)
2.	Complexity	Medium(3)
3.	Add-on Function	Very Low(1)
4.	Necessary Function	Very High(8)
5.	Flexibility	Low(2)
6.	Modularization of task	Medium(3)
7.	Time to Market	High(5)
8.	Amount of Requirement Known Initially	High(5)
9.	Clarity and completeness of requirements	High(5)
10.	Expandability	High(5)
11.	Coupling	High(5)
12.	Business Risk	Very High(8)
13.	Technical Risk	High(5)
14.	Operational Risk	High(5)
15.	Programmer's Capability	Very High(8)
16.	Application Experience	Low(2)
17.	Reuse of existing code	Very Low(1)
18.	develop for reuse	Medium(3)
19.	Platform volatility	Very Low(1)
20.	Platform experience	Low(2)
21.	Tool Experience	Medium(3)
22.	Team cohesion	High(5)

Input:- 1;3;1;8;2;3;5;5;5;5;5;8;5;5;8;2;1;3;1;2;3;5

Target:-2(i.e Both of them are suitable)

Output Screen:-



Chapter 5

Conclusion and Discussion

In this study we identified whether a project is fit for development using Agile Methodology or Traditional Methodology, or both. For this purpose, we identified 22 project metrics based on their impact on the selection of software development lifecycle. We assigned to these metrics, weights by taking into account their bias towards either Agile or Traditional Methodology. By examining a set of sample projects with our technique we came to the following conclusions,

1. The proposed technique is helpful in predicting which Methodology should be followed while developing a particular project. This can help organizations save on huge losses incurred upon failure of projects for selection of wrong process model.
2. The prediction process can be made more intelligent by use of machine learning algorithms. In our study we used Feed-Forward Back-Propagation neural network with 1 hidden layer. Different values of the output neuron give the class to which the project belongs.

Based on our experience while working on this problem we have come across some points that could be used as basis for further exploration in this area,

1. The model has been validated on some particular type of project data. We need to run this model on various other types of project data. Such validation is important for industrial application of this method.
2. We have used only one machine learning algorithm in this study. This does not ensure proper selection of machine learning algorithm for this kind of data. Algorithms like Random Forest, Logistic Regression, Decision Tree, Bagging, etc. should also be tried over datasets to select the best algorithm for such classification.

3. We have not developed any tool for the framework given in 3.1. Therefore, we do not have any repository for storage of the samples. This tool will make the lifecycle selection task more robust and easy.

PUBLICATIONS

During the period of working over this project we interacted with International community working on software engineering. Our Research papers have been accepted in International conference for presentation and will be published in their proceedings.

This paper presents the concept of applicability of agile methodology for mobile software development and proposed a technique for domain specific (various age group) priority based implementation of mobile services. In this paper, we also discussed the applicability of method configuration for mobile domain.

6.1 The details of Conference publications:

Conference Name: *International Conference on Software Engineering and Research Practices (SERP-12), Las Vegas, USA.*

URL: http://www.world-academy-of-science.org/

Paper Title: *"**Domain specific priority based implementation of mobile services- an agile way**"*

URL: http://www.ucmss.com/main/papersNew/papersAll/SER2782.pdf

Authors: Dr. Daya Gupta, Rinky Dwivedi, Sinjan Kumars

Location: Monte Carlo Resort, Las Vegas, Nevada, USA

Publishers/ proceedings: The accepted papers will be published in printed conference books/proceedings (they will also be available on the web). The proceedings will be processed for indexing into science citation databases that track citation frequency/data for each paper. These science citation databases include: Inspec / IET / The Institute for Engineering and Technology, CiteSeerX citation index, Google Scholar, Microsoft Academic Search, and other science databases. Like prior years, extended versions of selected papers (about 40%) will appear in journals and edited research books (publishers include: Springer, Elsevier).

References and Bibliography

[1] Kruchten, P., The Rational Unified Process, 2nd ed: Addison Wesley, 2001.

[2] B. Boehm and R. Turner, *Balancing agility and discipline: A guide for the perplexed*, Addison-Wesley, 2003.

[3] Brinkkemper. S. (1996) *Method Engineering-Engineering of Information Systems development Methods and Tools*, in Information and software technology,38,pp.275-280 (1996).

[4] Shyam R. Chidamber and Chris F. Kemerer," A metrics suite for object oriented design," IEEE transactions on software engineering, vol. 20, no. 6, June 1994.

[5] Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective Martin Hitz and Behzad Montazeri. IEEE transactions on software engineering, vol. 22, no. 4, april 1996.

[6] Henderson-Sellers, B. *et al*, (2002) *Using OPEN's deontic matrices for e-business*, in: C. Rolland, S. Brinkkemper, M.Saeki(Eds.), Engineering Information Systems in the Internet Context, Kluwer Academic Publishers, Boston, USA, 2002, pp. 9-30.

[7] Nielsen, J.: "Agile Development Projects and Usability" http://www.useit.com/alertbox/agile-methods.html

[8] Gupta D. and Prakash N. (2001) *Engineering Methods from their Requirements Specification*, in Requirements Engineering Journal 2001, 3, pp.133 – 160.

[9] Harmsen, A.F., Brinkkemper, S., Oei, H. (1994) *Situational Method Engineering for Information Systems Projects*. In Methods and Associated Tools for the Information Systems Life Cycle. Proceedings of the IFIP WG8.1 Working Conference Cris/94, T.W. Olle, A.A. Verrijn-Stuart, Eds. North Holland, Amsterdam, 1994, 169-194.

[10] Harmsen F, Brinkkemper S.(1995) *Description and manipulation of method fragments for method assembly*. In Proceedings of the workshop on management of software projects. Pergamon Press, London, 1995.

[11] Dias, M. V.: A new project management approach for software development. Master's Thesis, Universidade de São Paulo (2006) http://www.teses.usp.br/teses/disponiveis/12/12139/tde- 03012006-122134/

[12] J. Coplien, D. Hoffman, D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software*, pp. 37-45, 1998.

[13] A Conceptual Knowledge Base Representation for Agile Design of Human-Computer Interface(2009 Third International Symposium on Intelligent Information Technology Application) IEEE .

[14] Aggarwal K.K., Singh Y. ,Software Engineering, 2nd Edition, New Age International Publisher.

[15] Sommerville I., Software Engineering -8th Edition

[16] E. Valavanis, C. Ververidis, M. Vazirgianis, G.C. Polyzos, K. Norvag,"MobiShare, Sharing Context-Dependent Data & Services from Mobile Sources", in *Proc. of the IEEE/WIC International Conference on Web Intelligence (WI 2003)*, 2003.

[17] M. Haahr, R. Cunningham, V. Cahill, "Towards a Generic Architecture for Mobile Object-Oriented Applications", in *Proc. of the 2000 IEEE Workshop on Service Portability and Virtual Customer Environments*,2000.

[18] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M.Korkala, J. Koskela, P. Kyllönen, and O. Salo, "Mobile-D: An Agile Approach for Mobile Application Development", *in Proc of the OOPSLA'04 Conference*, 2004.

[19] Kelly, S., Lyytinen, K., Rossi, M.(1996) *Metaedit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment.* In Proceedings of the 8th Conf. on

Advanced Information Systems Engineering, Y. Vassiliou, J. Mylopoulos, Eds. Springer-Verlag, 1996.

[20] Boehm B. W. et al.,The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems.

[21] Boehm B.W., "Software Engineering Economics," Prentice Hall, 1981.

[22] Boehm B.W. et al , "Software Cost Estimation with COCOMO II," Prentice Hall, 2000.

[23] Ralyté, J. (2004) *Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks*, In Proceedings of the International Conference on Information Systems Development (ISD'04), Vilnius Technika, 2004, 271-282.

[24] Kemerer C.F., "An Empirical Validation of Software Cost Estimation Models," Comm. ACM, vol. 30, no. 5, 1987, pp. 416–429.

[25] Gupta D. and Prakash N. (2001) *Engineering Methods from their Requirements Specification*, in Requirements Engineering Journal 2001, 3, pp.133 – 160.

[26] Chen Z. et al, "Finding the Right Data for Software Cost Modelling".

[27] Boehm B. W., "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", "ACM", 11(4):14-24, August 1986.

[28] Rolland, C., Prakash, N.(1996) *A Proposal for Context-Specific Method Engineering*. In Method Engineering. Principles of Method Construction and Tool Support. Proceedings of IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering, 26-28 August 1996, Atlanta, USA, S. Brinkkemper, K. Lyytinen, R.J. Welke, Eds. Chapman & Hall, London, 191-208.

[29] Gupta D., Dwivedi R., Configuration from Situational Method Engineering, ACM SIGSOFT, Software Engineering Notes. Page 1, May 2012, Volume 37, Number 3.

[30] "Software Cost Estimation with COCOMO II", Prentice Hall, 2000.

[31] Beedle, M., et al., Principles behind the Agile manifesto, http://www.agilemanifesto.org/principles.html

[32] NguyenV.P. and Henderson-Sellers, B. (2003) OPENPC: A tool to automate aspects of method engineering, ICSSEA 2003. 16th International Conference on Software and Systems Engineering and their Applications, ICSSEA 2003, Paris, France, vol. 5, 7 pp.

[33] Weiss D.M., Lai C.T.R., "Software Product-line engineering: A Family based Sofwtare development Process," *Addison Wesley,1999.*

[34] Gupta D., Dwivedi R., "A Step towards Method Configuration From Situational Method Engineering", Software Engineering : An International Journal (SEIJ), Vol. 2, No. 1, March 2012.

[35] MATLAB, http://www.mathworks.com/

[36] www.agilealliance.org

[37] www.agilemethodology.org

[38] http://www.executivebrief.com/software-development/minimal-marketable-product.

[39] http://www.executivebrief.com/agile.