CHAPTER 1

INTRODUCTION

This research focuses to study, categorize and examine a number of methods for describing or modeling how software systems are developed. A study and analysis of traditional software life cycle models that dominate the field and current software development practices is taken care of which is, followed by a more comprehensive review of the models of software evolution that are of current use as the basis for organizing software engineering projects and technologies and a generalized model is obtained. The idea of a software process model that fits every project seems unlikely because any project has so many aspects that it is difficult to capture every potential aspect in a single perspective, this study helps to provide solutions in the form of a generalized model which produces reliable, correct and high quality software for the companies.

1.1 SOFTWARE ENGINEERING

- Software Engineering (SE) is a profession that is dedicated to designing, implementing, and modifying software in order to develop software that is of high quality, more affordable, maintainable, and fast to build. It is a systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of a software. The term software engineering first appeared in the 1968 NATO Software Engineering Conference to discuss software crisis. [1,2,31,33]
Software Engineering is said to be a *modeling activity* as it deals with complexity by focusing on the relevant details at a time.

- model – as an abstraction of reality
- analysis – by constructing a model of the problem domain
- design – by constructing a model of the solution domain

Since models are used to derive an acceptable solution, it is also considered as a *problem solving activity*.

- it is driven by experimentation
- it reuses pattern solutions
- incremental evolution of the system as acceptable to the client
- it is revised in response to change

It is also a *knowledge acquisition activity* as it involves data collection, organizing it into information and formalizing it into knowledge. It is nonlinear as any new information may invalidate previous knowledge.

- risk-based development – it identifies high-risk components
- issue-based development – it executes development activities in parallel, organizing according to issues which need resolution
- iterative development – it allows to design and implement the high-risk that is, the difficult parts first

It is also taken as a *rationale-driven activity* as the software engineers need to confine the context in which decisions were made and the underlying principle in order to understand the proposition of a proposed change when reviewing a decision.

- it assists in dealing with the changing systems
it is useful in the maintenance phase

Wasserman [32] categorizes eight fundamental notions that form the basis for an efficient discipline of software engineering:

- **Abstraction** – it is a description of the problem at some level of generalization that allows to concentrate on the major aspects of the problem without getting into the details

- **Analysis and Design Methods and Notations** - when working as a team, we must communicate with many other participants in the development process, and thus need a common way or method or notation for communication and documentation

- **Prototyping** – it is building a small version of a system, generally with limited functionality, this helps the user to identify the key requirements of a system and demonstrates the feasibility of a design or approach. It is used to design a good user interface

- **Software Architecture** – it is the description of a system in terms of a set of architectural units, followed by a map of how the units relate to one another
• **Software Process** – it is the organization of the activities of the process for developing software and to improve the quality of the software and the speed with which it is developed

• **Reuse** – it is taking advantage of the common functionality across applications by reusing items from the previous development

• **Measurement** – it is the quantitative descriptions of the improvements in processes, resources, and methods. It helps us to compare progress across disparate projects and support analysis and decision-making

• **Tools and Integrated Environment** – the computer-aided software engineering (CASE) tools are designed to enhance software development, but hardly address the entire software development life cycle

The causes of the software crisis were linked to the overall complexity of the hardware and the software development process. The crisis was related to the projects that were running over-budget, projects that were running over-time, inefficient softwares, softwares of low quality, softwares not meeting the requirements, unmanageable projects and projects where code was difficult to maintain, and softwares that were never delivered.

Problems that come across while developing the software and that are crucial in decision making are known as software crisis. This term is used to describe the impact
of rapid increases in computer power and the complexity of the problems, which could be tackled that is, it refers to the difficulty of writing correct, understandable, and verifiable computer programs. The base of the software crisis is complexity, expectations, and change. F. L. Bauer at the first NATO Software Engineering Conference coined the term software crisis in 1968 at Garmisch, Germany.

Crisis can arise due to problem in compatibility, portability, documentation, coordination of work or even poor maintenance of the software. Crisis can also be seen due to rise in the cost of the software, prices of hardware suddenly going down, problems associated with different versions of the software, to name the few. The causes of the software crisis are linked to the overall complexity of the software process. [3,4,5]

Booch defines the most visible symptoms of the software crisis as, [35]

- Late delivery, over budget
- Product does not meet specified requirements
- Inadequate documentation

Some observations on the software crisis are,

- A malady that has carried on this long must be called normal? (Booch, p. 8)
- Software system requirements are moving targets
- There may not be enough good developers around to create all the new software that users need
- A significant portion of developers and time must often be dedicated to the maintenance or preservation of geriatric software

The IEEE Computer Society's Software Engineering Body of Knowledge (SWEBOK) defines "software engineering" [2] as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of
software, and the study of these approaches. It is the application of Engineering to software as it integrates significant mathematics, computer science and practices whose origins are in Engineering. [6,7]

SWEBOK guide as per the 2004 edition, defines ten knowledge areas (KA) within the field of software engineering:

- Software requirements
- Software design
- Software construction
- Software testing
- Software maintenance
- Software configuration management
- Software engineering management
- Software engineering process
- Software engineering tools and methods
- Software quality

Software engineering applies technologies and practices from computer science, project management, and other fields. It improves the productivity of developers and the quality of the applications created. SE applications are used in a wide range of activities, from industry to entertainment. Software applications improve productivity and quality. Some of the application software examples are office suites, video games, and the World Wide Web and system software examples are embedded systems and operating systems. Software engineering is related to databases, languages, libraries, patterns, and tools. Computer science is related to algorithms and data structures, and project management is related to processes.
The SE community includes 630,000 practitioners and educators in the U.S. and an estimated 1,400,000 practitioners in the E.U., Asia, and elsewhere; it is about 60% the size of traditional engineering. SE pioneers include Barry Boehm, Fred Brooks, C. A. R. Hoare, and David Parnas. [8,9]. Software is a set of instructions, which acquires input and manipulates them to produce desired outputs. Computer software, or software is a general term used to describe a collection of computer programs, procedures and documentation that perform some tasks on a computer system.

Software is logical rather than a physical system element. Software is described by its capabilities. By capabilities we mean the features, functions and facilities provided by the software. The software is developed keeping in mind the hardware and operating system i.e. the platform to be used by the software. Software is the collection of computer programs, procedure rules, associated documentation and data. Software means a collection of programs whose objective is to enhance the capabilities of the hardware to meet out the objectives.

Software engineering is an engineering approach to developing software and is based on well-experimented principles. It is a systematic and disciplined approach for developing and maintaining software that is reliable and works efficiently in real world. It is concerned with all aspects of software production. Programs are developed by individuals for their personal use and are usually small in size and have limited functionality. Proper use of documentation and good user interface are generally missing. [12,13,14]

Software engineering is the field of computer science that deals with the construction of large or complex software system. A group of software professionals rather than a single professional is involved in the software development activity. The field computer science is concerned with theory and fundamentals, whereas software
engineering is concerned with the feasibility of developing and delivering useful software.

The development of large software is a group activity or it can be termed as software engineering activity. A software component or utility written at one source can be combined with components written at another source to build a system. Others can modify the components and it can be used to construct different versions of the system. The difference between programming and software engineering activity is that programming is primarily a personal activity, while software engineering is basically a team activity.

As per IEEE definition, Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software. It can also be defined as a discipline whose aim is to produce error free software that satisfies the user’s requirements and can be delivered on time within the budget. Software engineering is an engineering discipline, which is concerned with all aspects of software production. It is said to be a methodological and managerial discipline concerning the systematic production and maintenance of software products that are developed and maintained within the anticipated controlled time and cost limits.

1.2 SOFTWARE ENGINEERING GOALS AND PRACTICES

Today the modern software systems are much larger and more complex than the applications of the past. Older programming techniques are inadequate to implement the requirements of the newer systems. Software Engineering was born to cater to the needs of the sophisticated programming techniques. Generally practices such as top-down design, structured programming and design, pseudocode with iterative
refinement, walk-throughs and OOP (Object Oriented Programming) are considered to be part of this discipline. Many advanced constructs in modern programming languages such as C++ were incorporated to meet the Software Engineering Goals given below. [34]

1. All programming projects should begin with functional descriptions, which becomes the highest-level pseudocode. Every module and every routine, high or low level should have a functional description appropriate to its level of abstraction.

2. Functional descriptions determine that high-level pseudocode should be refined to the low level pseudocode. The pseudocode becomes an internal documentation.

3. Pseudocode should be organized into modules, thus ultimately organizing the project into modules.

4. Computer code for all routines and main programs should be generated from low-level pseudocode. The pseudocode should become internal documentation.

5. Module interface specifications should be created so that modules can be created independently.

6. Programs should be constructed from the designed modules.

7. Support routines should reside in separately compiled modules.

8. Each module should contain only those routines whose functions are related.

9. Inter-module communication should be accomplished only by passing parameters. No global variables should be used. Parameters should be passed "by value" where required access is read-only.
10. Communication between routines in the same module should also be accomplished only by parameter passing.

11. Low-level routines should have simple and explicit functions.

12. Support routines should be evaluated with respect to flexibility and the efficiency.

13. Include files should be used where required to effect inter-module independence and consistency.

14. Changes in low-level routines should not require source code changes in high-level routines.

15. Changes in high-level routines should not require source code changes in low-level routines.

16. Changes to high-level routines should not require low-level routines to be recompiled.

17. Changes to low-level routines should not require high-level routines to be recompiled.

18. Modules should be constructed so that private support routines are not accessible outside the module. This can be accomplished using sub modules.

19. Modules should be constructed so that internal data structures can be accessed only through the authorized routines.

20. All program changes should start with a change to the highest-level pseudocode appropriate to that change. This will ensure that pseudocode accurately reflects the structure of the underlying code.
Software products have multiple users, good user interface, user’s manual and good documentation support. Software engineering deals with cost effective solutions to solve the problems practically. Software engineers develop software products by adopting well-defined and systematic development methodologies. [11]

The performance goals of software engineering are usually taken as follows:

• Maintainability – the ability to easily make changes, enhancements or improvements

• Dependability – the ability to rely on the software to function properly when needed

• Efficiency – the ability of software to use computing resources effectively (essentially the attributes like, space and time)

• Usability – the ability for the end user to easily and effectively put the software to proper use

Software Engineering Practices

Software engineering practices are difficult to be defined accurately. Each has utility in various environments; a brief description of each is given as below. [34]

OOP (Object Oriented Programming) and Abstract Data Types

Many applications like database application are data-driven. For such, applications it is important to define classes of data objects. Most data elements can be defined as classes. Objects are instances of a class. The particular attributes that define a class are application dependent. After defining a class it is normal to add methods
(functions) to the class. The class together with its methods is often referred to as an ADT or Abstract Data Type.

**Top-down design**
Top-down design is a process by which we iteratively that is, in stages define the functional requirements of a complex system with more detailed specifications.

**Pseudocode**
Pseudocode is a natural-language-like statements that define functions to be performed. Iterative refinement of pseudocode refers to the process of successively expanding the statements to create more detailed descriptions of function definition.

**Structured programming and design**
The structure of a complex system may be broken down into smaller parts with their own functional descriptions. These smaller components are divided into libraries of related functions. These functions are then tested independently and reused in many other programs. This provides an efficient methodology for many programmers to work on a single project and the same computer code can be reused on many programming projects. This reduces the time required to implement a system, reduces errors in the system, reduces the cost per line of code, and increases programmer productivity and utilization.

**Bottom-up implementation**
A system can be constructed from its component parts (classes) and functions. These component building blocks might have been already constructed and tested during the construction of a previous project.

**Structured testing**
Computer professionals consider it irresponsible to release program code without thoroughly testing it. Structured testing makes every effort to provide a formal model
for identifying functional areas of a program and execution paths that should be tested and for providing data to accomplish the testing.

**Structured walk-throughs**
It refers to a methodical and complete construction of a plan for simulating the running of a system before it is implemented so that it may ascertain whether the system is functioning according to the system specifications.

**Encapsulation**
Definitions and methods performed on instances of data are encapsulated into objects.

**Reusable code**
It simply refers to the goal of writing code in a flexible manner so that it may be used many times and preferably be able to be reused in different environments. Ideally, the reusable code is in the form of a function object library. Application programmer should only need to refer to the functional description of the functions and the calling interface. Implementation details should be irrelevant and source code need not be inspected in order to use the functions. This depends only on the functional description and calling interface and is also referred to as Information Hiding.

1.3 SOFTWARE PROCESS

A software process defines the systematic approach that is used as software is engineered. Software engineering is performed by creative and knowledgeable people. A software development process is used to develop computer software and it refers to a standard methodology which has been used on similar projects earlier, or, which is used consistently within an organization.

Researchers and practitioners have realized that software development is a collective, complex, and creative effort and that developing software is not just creating effective
programming languages and tools. The quality of a software product mainly depends on the people, organization, and procedures used to create and deliver it. This vision was developed from the work accomplished during the 60s and 70s. In those two decades, researchers and practitioners focused their activity on three main goals:

- Development of structured programming languages
  (e.g., Algol, Pascal, and C).

- Development of design methods and principles
  (e.g., information hiding, top-down refinement, functional decomposition).

- Definition of software lifecycles
  (e.g., waterfall, incremental development, prototype-based).

A software lifecycle defines different stages in the lifetime of a software product. Generally, they are requirements analysis and specification, design, development, verification and validation, deployment and maintenance. Additionally, a software lifecycle defines the principles and guidelines according to which these different stages have to be carried out.

A software process can be defined as the consistent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product. Thus, a software process exploits a number of contributions and concepts:

1. *Software development technology*: it deals with the technological support used in the process. To accomplish software development activities we need tools, infrastructures, and environments and a proper technology that makes it possible and economically feasible to create the complex software products.
2. **Software development methods and techniques**: these are the guidelines on how to use technology and accomplish software development activities. For effective usage of technology the methodological support is essential.

3. **Organizational behavior**: it is related to the organizational structure. As teams of people carry out the software development, they have to be coordinated and managed within an effective organizational structure. [17,18,37]

It is a systematic application of scientific and technological knowledge, using sound engineering principles, to produce computer programs, requirements definition, functional specification, design description, program implementation, and test methods that make up a code. A software development process is also known as a **software development lifecycle** and is a structure imposed on the development of a software product. Several models for processes exists, each describing various approaches to different tasks or activities that take place during the process. [16,19,20,21]

Software process is a structured set of activities required to develop a software system which includes, specification, design, validation and evolution.

### 1.4 SOFTWARE PROCESS MODELING

Models of software evolution dates back to 1950's and 1960's (Hosier 1961, Royce 1970). The purpose of early software life cycle models was to provide a conceptual scheme for rationally managing the development of software systems. They serve as a basis for planning, organizing, staffing, coordinating, budgeting, and directing software development activities. Since 1960's, many descriptions of the classic software life cycle have appeared (e.g., Hosier 1961, Royce 1970, Boehm 1976,

The classic software life cycle models usually include the following basic activities for the development of software: [15]

**System Initiation/Planning:** it is the starting point of the system

**Requirement Analysis and Specification:** it identifies the problems a new software system is to solve, its operational capabilities, its desired performance characteristics, and the resource infrastructure which is needed to support system operation and maintenance

**Functional Specification or Prototyping:** identifies and formalizes the objects, their attributes and relationships, the operations, the constraints etc.

**Architectural Design and Configuration Specification:** it defines the interconnection and resource interfaces between subsystems, components, and modules suitable for their detailed design and overall configuration management

**Detailed Design Specification:** it defines the procedural methods through which the data resources within the modules of a component are transformed from inputs into outputs

**Implementation and Debugging:** it codifies the specifications into source code and validates their basic operation

**Software Integration and Testing:** it confirms the overall integrity of the software system architectural configuration, verifying the resource interfaces and
interconnections against their specifications, and validating the performance of the system and subsystems against their requirements.

**Documentation Revision and System Delivery:** it is concerned with, wrapping system development descriptions into systematic documents and user guides, all in a form suitable for dissemination and system support.

**Deployment and Installation:** it includes, providing directions for installing the delivered software into the local computing environment, configuring operating systems parameters and user access privileges, and running diagnostic test cases to assure the feasibility of basic system operation.

**Training and Use:** it is concerned with providing system users with instructional aids and guidance for understanding the system's capabilities and limits in order to effectively use the system.

**Software Maintenance:** it is sustaining useful operation of a system in its host environment by providing requested functional enhancements, repairs, performance improvements, and conversions.

The basic techniques to model business processes are flow chart, functional flow block diagram, control flow diagram, Gantt chart, PERT (Program or Project Evaluation and Review Technique) diagram, and IDEF (Integration DEFINition).

The Gantt chart first arrived around the 1900, the flow charts in the 1920s, Functional Flow Block Diagram and PERT in the 1950s, Data Flow Diagrams and IDEF in the 1970s. Modern methods are Unified Modeling Language (UML) and Business Process Modeling Notation. The most popular UML modelling tool is IBM Rational
Rose. Other tools include Rational Rhapsody, MagicDraw UML, StarUML, ArgoUML, Umbrello, BOUML, PowerDesigner, and Dia. [23]

Software process model is an abstract representation of a software process and each process model represents a process from a particular perspective. Regardless of the process model that is selected, a generic process framework that encompasses the framework activities like, communication, planning, modeling, construction and deployment is available.

There are various software development approaches known as ‘software development process model’. A process model for software engineering is chosen based on the nature of the development project. [22, 24, 28]

Some of the traditional software models used in Software Engineering are:

- Linear Sequential Model or Waterfall Model or Classic Life Cycle
- Incremental Model
- Prototyping Model
- Rapid Application Development Model
- Spiral Model
- Concurrent Development Model
Some of the specialized process models used in Software Engineering are:

- Component-Based Development
- Formal Methods Model
- Aspect Oriented Software Development

Today, Unified Process is an attempt to use the features and characteristics of conventional software process models and implement them with the best protocols of agile software development.

Process models are the basic concepts in the software engineering. Process models are processes of the same type that are classified together to form a model. The same process model is used repeatedly for the development of many applications and has many instantiations. Use of a process model is to prescribe how things should be done. What the process shall be is determined during actual system development. [25]

The basic aim of a process model is to be:

- **Descriptive**
  - track what actually happens during a process
  - acquire point of view of an external observer who looks at the way a process has been performed and determines the improvements that must be made to make it perform more effectively or efficiently
• Prescriptive
  
  - define the desired processes and how they should be performed
  
  - establish rules, guidelines, and behavior patterns which would lead to the desired process performance

• Explanatory
  
  - provide explanations about the rationale of processes
  
  - explore and evaluate several possible course of action based on rational arguments
  
  - establish explicit link between processes and requirements that the model needs to fulfill
  
  - pre-define points at which data can be extracted for reporting purposes

From a theoretical point of view, the meta-process modeling explains the key concepts needed to describe what happens in the development process, on what, when it happens, and why. From an operational point of view, the meta-process modeling is aimed at providing guidance for method engineers and application developers. [26]

The activity of modeling a process is related to the need to change the processes or to identify issues which needs to be corrected. Change management programmes are desired to put the processes into practice. Supporting technologies include Unified
Modeling Language (UML), model-driven architecture, and service-oriented architecture (SOA).

Process modeling addresses the process aspects of an Enterprise Business Architecture, leading to an Enterprise Architecture. Process modeling has always been a key aspect of the business process reengineering, and continuous improvement approaches as seen in Six Sigma.

1.4.1 Advantages of implementing software engineering process models

The North Atlantic Treaty Organization Science Committee had discussions on the topic concerning the state of Computer Science. There were worldwide issues with the development of software, the crisis being that software projects did not seem ever to complete. The study group coined the term "software engineering" to be provocative and implying need for software manufacturing to be similar to traditional branches of engineering. In the beginning, individual programmers used whatever means worked to build software. Formal methods of design or programming did not exist. Programmers were never able to give a definitive estimate as to how long a project would take. Software projects often were behind schedule, over cost, poorly documented, poorly designed, and contained items other than the requirements. These projects were costing corporations thousands of dollars. The software industry was quite undisciplined and it was obvious. [27, 29] With the implementation of the process models an organized way is developed which results in the cost effective and time oriented development. The productivity increases with the reduction in production time. This results in an increased customer satisfaction and satisfied employees. [30,38]

It is always critical for managers and analysts to ensure that their processes are optimized and are running smoothly:
Process modeling allows analysis and understanding of the process flows and helps to know if there is room for further optimization and efficiencies.

It helps to make improvements and reduce process cycle time.

It increases productivity of existing resources and staff thus making the team to do more with less effort.

It facilitates risk free experimentation and encourages exchange of process improvement ideas.

It allows modeling and simulation of process designs before actually implementing them thus minimizing disruptions.

It encourages a mind-set of continually optimizing business critical processes to incrementally improve operational efficiencies.

It enables better resource utilization

1.5 THESIS OVERVIEW

The objective of this research is to study, categorize and examine various methods for describing or modeling, as how software systems are developed and to provide solutions in the form of a generalized model which help in producing a correct and high quality software for the companies.
A study and analysis of traditional software life cycle models that dominate the field and current software development practices is conducted. It is followed by a more comprehensive review of the traditional models of software evolution that are of significant use and are considered as the basis for organizing software engineering projects and technologies. The idea of a software process model that fits every project seems unlikely because every project has so many aspects that it is difficult to capture every potential aspect in a single perspective, the study helped to provide solutions in the form of a generalized model which may help in producing a correct and high quality software for the companies.

In Chapter 1, we present four primary objectives. The first is to explain the concept of software engineering, it’s primary role as a part of software development process and it’s purpose in the field of software development. The second objective is to explain the software process as a systematic and disciplined approach to the development and maintenance of the software and the methodology it uses to provide systematic, cost effective and reliable way to develop software. The third objective is to explain and describe the basic activities for the development of software, which are used in all the traditional software process models, which serve as the basis of any new process model developed. Lastly the fourth objective is to explain software process modeling and its aim along with the advantages of implementing software engineering process models.

In Chapter 2, we provide a general evaluation objective, which means to appraise or assess the significance of evaluation. It includes the general evaluation objectives, which take into account a general understanding of the current practices, confirming the theories or conventional knowledge, exploring when a domain is not well understood and describing the current state of things. Evaluation helps in predicting the future and explaining why the things or sequences are taking place.
In software engineering, for evaluation measures we need to understand the software process and the product. Various approaches to perform the software evaluation are discussed in this chapter. Methods for model definition are also discussed. It explains various types of customers for whom the software is developed followed by the methods for data capture. The existing methods for analysis are also described. Further, it discusses the verification and validation techniques for managing the validity treats. The research methods designed for objective evaluation are stated considering that evaluation can be conducted using construct validity, internal validity, external validity and reliability.

In the next section the evaluation process is covered which consists of three stages namely, the quality refinement definition, evaluation preparation and evaluation procedure. Next, methods for solving a model are described which includes how to solve a model analytically, where a complete mathematical description of the model is required. If an analytical solution of the model is not feasible due to the limitations of the modeling formalism or model complexity, or not desirable, another way, by means of using computer based simulation is described in which one may approximate the behavior of the real system by executing the model over time, and subsequently or interactively draw conclusions about reality from the observed dynamic behavior of the model.

In Chapter 3, the software process is revisited from the point of view that there exists a need for a generalized model after studying the basis of process model development. The factors that influence the process models and enhance the capabilities of the processes are described. The fundamental design concepts, which have evolved, are discussed here.

Structured planning and execution of the activities is required for model development and simulation. It is called the Model Development and Execution Process, which is
described in the subsequent section. Modeling activities and intermediate products describe the activities carried out during model development, which can be organized in phases, according to their contents and chronological occurrence. Numerous model development processes exists, which can be distinguished by the selection of phases and the way in which the feedback between the phases is organized. It is also discussed that not only the iteration of development phases, but also the resolution or level of detail of the activity descriptions making up each phase can vary and it mainly depends on the architect of the model development process and the aspects of model development that seem to be most important to the particular group. Several classes of process models for the development of models, software products, or simulation results are studied, namely, Waterfall processes, V-Type processes, Lifecycle processes and Spiral processes. It is followed by design considerations. There are many aspects to consider in the design of software. The model design should be strong enough to contain the basic considerations while developing a system. The importance of each aspect should be taken care of which would reflect the goals the software, which are tried to achieve. Some of these important aspects are considered in this chapter.

In the next section Software Design Description are described. We present the IEEE 1016-1998, which is also known as the Recommended Practice for Software Design Descriptions,. This IEEE standard specifies an organizational structure for a Software Design Description (SDD).

In Chapter 4, we review the literature and present a brief summary of the work relating to the general process model. Here, we describe why we need the GSPM-Generalized Software Process Model. It affirms that the need to obtain a general model arises as the language independent process can be established and the techniques can then be designed that guarantee the conservation of model correctness during configuration, thus liberating analysts from using manual methods to check the
model correctness. Model should be able to eliminate redundancies in the process and facilitate standardization and reuse. Thus it is tried to develop an integrated framework or system to manage variability in process aware information system.

Study and comparison of the process models are performed to ensure the need for obtaining a generalized model. The traditional software lifecycle models are described and also as to where such models are needed, that is, in which situation such models are beneficial. A comparison table is also drawn for the traditional models.

Next, the shortcomings of various models are discussed. This is followed by an efficient paradigm for implementation of process models in projects. Initially, a model has been designed and proposed for the projects with flexible requirements. This is tested by showing the implementation in of the proposed model in the courseware development activity / project. Followed by is a new proposal that has been developed for the projects with dynamic and collaborative requirements. This model is further evaluated for its functioning.

In Chapter 5, we present the proposed work with the new concepts so as to eliminate the shortcomings and introduce new method for taking decisions.

The Generalized Model design is discussed and presented which is modeled for the optimization of the process which will help the software development team to work with larger efficiency and effectiveness. It is followed by the basic structural model designed for it. Next is the data flow diagram of the GSPM followed by the internal structure of GSPM. For a process model to be successful it is essential that confirmation of the activities is performed along with the participation of the stakeholders. Once the confirmation is sought the activity of code design and development should take place. The re-use of the components acts as the basis of reducing time factor in the development phase. The organization of the module should
be such that it should be easy to modify. Execution of the process should be easy to measure and it should be possible to refine that execution. Model should be flexible so that it can provide guidelines on the execution and tracking of a project. Thus emphasis has been on productivity improvement and reuse building. Effective testing and continuous validation from customer side will enhance confidence in services being provided making the system more stable.

The case study performed on the e governance is described and the benefits are listed. Another case study is performed on Student Information System, a project carried out at Birla Institute of Technology, Mesra, Ranchi, Extension Centre, Jaipur. The comparison is made between the project using the mostly used Spiral methodology and the proposed methodology, GSPM. Thereby depicting the effectiveness and usefulness of the proposed model.

In Chapter 6, based on the case study, an analysis is drawn using the mathematical modeling language, Petri net, which is a directed bipartite graph offering graphical notation for stepwise processes. Bar charts indicating the number of inputs, number of function points and number of iterations are drawn for easy comparison of the existing and the proposed Generalized Software Process Model [GSPM]. This chapter states the advantages of using the proposed model. We demonstrate the applicability of the proposed approach by using Petri nets for all the traditional models and the proposed model. It is followed by the graphical analysis based on the number of input points, number of function points and number of iterations, which are the basis of model execution and performance checking.

The thesis ends with our final analysis and conclusion.