EVOLUTION AND MELIORATION OF SOFTWARE MANAGEMENT PROCESSES

Sunil Kumar Jangir[1], Neha Gupta[2], Shreya Agrawal[3]

[1]Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur

sunil.jangir07@gmail.com

[2]Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur

it.nehagupta@gmail.com

[3]Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur

shreyaagrawal.it.jecrc@gmail.com

ABSTRACT

Evolution of software engineering practices consecrates a novel glide over on the progression of software from an elementary form to a more intricate and highly specialized one. As the current software-development process faces conflicting demands and as the productivity remains quite unimpressive, there is a conspicuous need to search, cull down and adopt the best practices in the industry. Practices which have been indelibly commingled with its long history and practices, which define business requirements - requirements that act as a conduit between the user needs and capabilities of software technologies. This beckons for an optimized implementation of successive software practices suggested by perspective software models and delineate methods for scrutinizing associated risks. Thus, this paper focuses on software evolution, software engineering practices, the need for improving a software process and measures to do the same.

KEYWORDS

Software Engineering Practices, Evolution, Best Practices

1. INTRODUCTION

Software is easy to change and this characteristic of software makes it the target for organizations. Software developers find the need to change the software throughout its entire development as well as after its delivery to the customer. These changes are based on changing or evolving requirements. It is hard to assess the impact of the change without actually implementing the change. Out of the percentage of damage done on the software, most is due to the frequent changes done to the software. Further, software development and its environment is itself changing at a very rapid rate that it becomes difficult to train and re – train the software engineers and invest so much of money in training.

The software-development industry has become an increasingly important business sector that also bears a notable impact on the remainder of the economy [87-90]. It had tried to follow the same path as other engineering disciplines but it had backed the wrong horse, resulting into its failure, as all engineering disciplines are some way or the other different from each other.

Around 50 years back, scientists performed most of the programming from solving small mathematical problems to making small snippets of codes. “Today, software is working both explicitly and behind the scenes in virtually all aspects of our lives, including the critical systems that affect our health and well-being [11, 79].” It is very spectacular to see the rapid progression of the software systems from a relatively small and less complex system to a huge, monstrous, and complex system. Fred Brooks said that:

Software systems are perhaps the most intricate and complex . . . of the things humanity makes [11, 80].

Rapid growth of the sector, increasing economic importance connected with the special characteristics of software itself and the dynamics of the software industry make software’s technological, process, and business aspects an interesting and challenging area to study [87, 90, 91].Despite the rapid progression of the software system, the software industry is considered to be in crisis. This is named as “software crisis”, a term coined in 1968 at a conference attended mostly by theorists rather than practitioners describes the inability of the software people to deliver quality assured software on schedule to the customers. It meant that the practice of software was in real crisis --- that is the projects where behind schedule, overly budgeted, and unreliable. This is also evident from various head-wringing articles about software projects and their failures. According to Gibbs, “the average software development project overshoots its schedule by half; larger projects generally do worse. And, some three quarters of all large systems are “operating failures that either do not function as intended or are not used at all [11, 81].” However, according to Glass, software crisis marked the beginning of the Golden Age of Computing Practice, which persists today. Nevertheless, the notion of software crisis reached a fever pitch of intensity in the 1980s and 1990s [82].

The world market for computer-based applications is worth hundreds of billions USD and affects almost all people’s life directly or indirectly [1]. The software to be developed is expensive and is a major cost factor in corporate information systems budgets. The magnitude of software investments, estimated at more than $200 billion annually [2, 3], force management to carefully consider the costs and benefits before committing the required resources to any potential software development project. Naturally, the accuracy of software project estimates has a direct and significant impact on the quality of the firm's software investment decisions [3]. In addition, software project failures can sometimes be catastrophic. Therefore, it is necessary to have qualified software professionals in the software-development area. The frailty of software developers to deliver coherent and efficient software gave emergence to the conspicuous need of ameliorating the software through evolution of software engineering practices and processes and adoption of the best software engineering practices in the industry. In addition, software-development is a not simple task [4] especially for large projects where big teams, often geographically distributed, are involved in [5].

In 2002, according to Voas, 40 percent of a typical corporation’s IT portfolio will be Commercial Of-the-Shelf Components (COTS) software [92].

In Section 1, we have discussed about the literature of software engineering and the challenges faced by it. Then, we move to Section 2 where we discuss about software process evolution and its various stages. In Section 3, we illustrate software engineering practices and the type of practices. Next in Section 4, we discuss about software processes. In Section 5, we throw some light on points necessary to improve the software process as a whole.

1.1. Software Engineering

Software Engineering is one of the most critical areas of Computer Science at academics and industrial levels. The term software engineering first appeared in the late 1960's to describe ways to develop, manage, and maintain software so that the resulting products are reliable, correct, efficient, and flexible [6].

1.1.1. Literature

Software engineering is arguably less than four decades old. Practitioners have been developing software for longer than that, of course. Land traces that history back to the early 1950s but in academe, software engineering is a somewhat newer field [35,36]. Its first conferences were held in the late 1960s, and its academic presence did not begin to separate off from computer science until the early 1980s [35].

Over the years, software engineering (SE) research has been criticized from several different points of view - that it is immature [29,35], that it lacks important elements such as evaluation [30,31,35] that is unscientific in its approaches [32,35]. There have even been attacked on the very foundations of SE research – that it advocates more than it evaluates [33, 35]; that it is, in fact, an endeavor in crisis [34, 35]. Most of these criticisms and attacks have been supported by appropriate research. For example, claims of immaturity are accompanied by a deep analysis of the progress made by more mature research fields; claims of failure to evaluate are accompanied by analysis of the relevant literature to see if software engineering papers include an evaluative component; and claims of advocacy are accompanied by at least quotations from papers that do precisely that [35].

In 1967, "the phrase software engineering" was chosen as being provocative [7]. Chaired by Fritz Bauer, the NATO science committee organized the first software engineering conference in 1968 at Garmisch, Germany [8]. Twenty-three years later, Mary Shaw of Carnegie Mellon University (CMU), having served as the Chief Scientist of Software Engineering Institute at CMU for several years (84-87), published a research paper Prospect for an Engineering Discipline of Software. The article was nominated as one of the most influential papers in 25 years since Software magazine's inception. In it, she wrote, “software engineering is not yet a true engineering discipline, but it has the potential to become one. Older engineering fields suggest the character software engineering might have [9, 35]." She predicts that the maturity of SE will depend on a number of things, including evolving ‘professional specializations’ and ‘improving the coupling between science and commercial practice [35].’ Mary Shaw in her paper, “The coming age of software architecture research,” provides a summary of the software engineering research state, which she calls a ‘challenge to the whole software engineering [29,35].’

Software engineering does not yet have a widely recognized and widely appreciated set of research paradigms in the way that other parts of computer science do. That is, we don’t recognize what our research strategies are and how they establish their results [35].

Barry Boehm in his keynote address, "A View of 20th and 21st Century Software Engineering” gave his definition of software engineering as, "*the application of science and mathematics by which the properties of software are made useful to people*."In this definition, he added, science should be regarded as computer science, as well as other sciences including behavioral sciences, economics, and management sciences [10]. Various people also defined software engineering in other ways. Manfred Broy defines engineering as a discipline and a profession with applying scientific knowledge and utilizing natural laws and physical resources for designing and implementing materials, structures, machines, devices, systems and processes that fulfil a desired objective and meet specified criteria. Then, he specifies the software engineering with: "*Applying scientific knowledge and utilizing the laws of informatics and application domains and computational and human resources in order to design and implement structures, machines, devices, systems, and processes that realize an objective and meet specified criteria [6].*" Broy categorizes the science of software engineering into three aspects: (i) Mathematics, including Logic and Discrete Mathematics, (ii) Software engineering principles, process, quality, economy, and tooling and (iii) Software infrastructure, and hardware.

Software engineering achieved popularity during 70s. As a discipline, software engineering has progressed very far in a short period, particularly when compared to a classical engineering field (like civil or electrical engineering). Software engineering is concerned with all aspects of software production from the early stages of system specification to maintain the system after it has gone into use [11]. Today's software engineering differs significantly from the classical forms of engineering in terms of process management, software tooling, and design activities for software development. Nowadays, medical sciences also use software engineering, for example, in dialysis of the human body.

In 2007 it was felt tha*t* the models available today have limited the present growth scenario of software engineering. Hence, a need to transform the software engineering model had been felt. This transformation was necessary to increase the efficiency of the processes in consideration [93, 96].

1.1.2. Principles

- Abstraction
- Modularity
- Iterative Enhancement
- Consistency
- Generality

1.1.3. Objectives

Four main objectives of software engineering are:

- Changeability
- Efficiency
- Comprehensibility
- Reliability

1.2. Milestones Around Software Engineering's Neck

Over the years, the term software engineering has been attributed with a number of definitions. A common one that is used to characterize the discipline is a methodical, engineering approach to the design and development of software production, throughout its whole life cycle [83]. But this discipline has been faced with a number of challenges over the years, including those related to requirements and schedule. Three fundamental challenges are:

- **Requirement Changes**

Unfortunately, requirement changes originate from various sources. Requirement analysts are not able to understand the product domain for implementing in the early product life cycle, as sometimes the customers are unable to express their share of requirements. This happens because often they themselves do not know what they require until they see some prototype or representation of what they intend. Secondly, the product domain may also change, new technologies maybe available in the market or the competitors may have already developed the same product with new features. All these points beckon for changing the product's software, thus, challenging software engineering.

- **Optimism of the crew**

The software engineers are responsible for estimating how long it will take to develop a product. No matter how many times the software-development crew has failed to deliver the product on the committed schedule, they never lose their optimism about delivering the product on schedule the next time. Consequently, they end up committing to a date, which is not feasible.

- **Pressure on the crew**

Often the crew is pressurized to make faster delivery. Hence, they end up in making aggressive commitments which results in their inability to deliver the product on time, thus retarding their reputation and growth.

2. SOFTWARE PROCESS EVOLUTION

Software process evolution is a term used in software engineering to develop software and then iteratively add new methods and update the existing ones to improve software engineering processes. For software to evolve, hundreds of developers perform more than one million changes over more than 6 years. The society increasingly relies on software but the software is unreliable and of low quality. It is regarded as a classical engineering product but is more complex than any other human artifact [15]. Thus, it is necessary to constantly update and improve our software that is software need to be evolved. In general, software evolves due to various reasons, such as changing requirements, new features, bug fixing or non-functional issues. During these engagements the architecture should remain evolvable, as this is the only way to avoid the system turning into a legacy system [53, 54], which is often called architectural drift or decay. In general, evolution is "*the assimilation of changes through generations of organisms that results in the origin of new species*." Systems that do not change are considered as dead processes. The evolution of software systems is usually thought of in terms of the kinds of changes that are made. While the overall motivation of evolution is adaptation, software changes usually partitioned into three general classes: corrections, improvements, and enhancements. Corrections tend to be fixes of coding errors, but may also range over design, architecture and requirements errors. Improvements tend to be things like increases in performance, usability, maintainability, and so forth. Enhancements are new features or functions that are generally visible to the users of the system [17]. Software engineering approaches often concentrate on initial software development and not on the continual evolution of the software and its environment. Software is continually changing and evolving, not only because of the discovery of latent errors, but primarily because of changes in the operating environment, in the needs of the end users, and in the underlying technology. The software must be designed to be changeable without compromising the confidence in the properties that were initially verified [16].

The term software evolution stems from a series of works, commonly referred to today as Lehman's laws, that were first proposed by Dr. Meir Lehman in his work Programs, Life Cycles, and Laws of Software Evolution (1980) [17]. Lehman's "*Laws of Software Evolution*," say that continuing change, increasing complexity and increasing size are the characteristics of software [18].

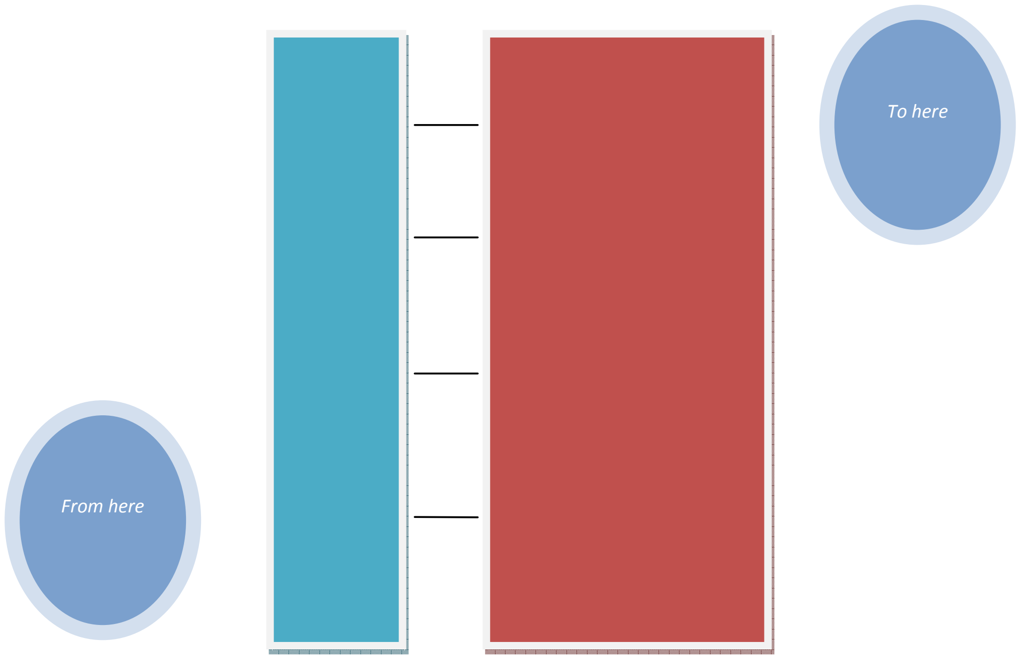


Figure 2. Software Evolution [18]

The figure above shows that as the software evolves its size also increases, which is according to Lehman's Laws of Software Evolution. That is when the software is in its initial stage (represented by blue bar) it is small, but as it evolves its size increases (represented by maroon block).

The most well-documented early attempt to study software evolution in a systematic way was conducted by Belady and Lehman beginning in the late 1960s [48, 49]. Their early collaboration continued to expand over the next decade [48-51], and resulted in a set of laws of software evolution [48]. In that seminal paper, Belady and Lehman outlined three laws of software evolution: (i) the law of continuous change, (ii) the law of increasing entropy, and (iii) the law of statistically smooth growth. In a later paper, Lehman revised the initial three laws and renamed them: (i) the law of continuing change, (ii) the law of increasing complexity (formerly the law of increasing entropy), and (iii) the law of self regulation (formerly the law of statistically smooth growth). In addition, he added two new laws, the law of conservation of organizational stability (also known as invariant work rate) and the law of conservation of familiarity [52]. These two additions describe limitations on software system growth.

Lehman and Belady's research found that once a module grew beyond a particular size such growth was accompanied by a growth in complexity and an increase in the probability of errors [51], which is also depicted in figure. 2. By the late 1990s, three additional laws of software evolution had been proposed: the law of continuing growth, the law of declining quality, and the feedback system [18, 49].

In 2000, Lehman in his paper, "Towards a Theory of Software Evolution- And its Practical Impact," said that evolution is generally supported by black box and white box studies of the FEAST projects.

2.1. How is software evolved?

A new approach to software evolution deals with the evolution of both: software processes and their models. These two types of evolution are complementary. The first one allows acting directly on the process during its execution. By appropriate update operations, it gives to the user the possibility to perfect his software process through simulation and to define new software processes dynamically. The second one allows evolving software process models to satisfy new software engineering requirements (new software component types, new tool types, new agent types, etc.) [19].

2.2. Stages of Software Evolution

Clusters of related software practices form evolutionary stages marked A to G.

2.2.1. Stage A – Review and Tracking

At this stage of software process evolution, the code and design are reviewed, and test trouble reports are tracked. Further, it controls the changes made to the code, design, and software requirements through configuration management, which is a newly added practice.

2.2.2 Stage B –Standardization of Processes

It aims at the standardization of the software-development process to ensure that the software organization uses a standardized and documented process. Consequently, before making any contractual commitments, the project managers' plan, organize and review the implemented procedures. One new practice of tracking software requirements and design is added. Another practice enables regular technical interchanges with the customer.

2.2.3. Stage C – Management of Reviews

Implementation of a higher level of review and change control mechanism is done in this stage. Review data are analyzed; action items are tracked to closure; code review standards are applied; configuration control is used for each project, and a formal software size estimation procedure is used [16]. Various computer tools are used for tracking and reporting the status of the software as a part of a new practice.

2.2.4. Stage D – Melioration of a Software Process

This stage represents the end-point of the middle evolution stages. Improvement of software-development processes is done in order to remove the inadequacies of prior implementation of the process and by checking its compliance with standard processes. This is done by adding new layers to the prior process without changing it.

2.2.5. Stage E – Test Coverage

This stage onwards, advanced engineering and management practices are gradually introduced. Test coverage for each phase of functional testing is done; measurement and coding of design for reuse and maintenance of data on planned and actual software units are also introduced.

2.2.6. Stage F – Scrutinize

Scrutinizing involves regression testing of the system. Regression testing is the testing in which new errors or regressions are uncovered in the existing functionalities after changes have been made to a system. An error database called metrics is maintained in order to keep track of the errors, their point of origin and to provide remedies.

2.2.7. Stage G – Modern and Sophisticated Practices

Establishment of a process measurement database and collection of data for projects is done. At this stage, a new practice of auditing the software process for each project is introduced. The second practice introduced is related to training programs aimed at preparing the organization.

Thus, the software process's evolution scale described here reflects the way software development and management practices are introduced in the industry, and how their use evolves "naturally" [20].

In a nutshell, the software evolution begins at Stage A, the lowest level, where the practice of reviewing code is considered. Simultaneously, code review standards are introduced after which test coverage is recorded to ensure universal application of the code reviews. Next code review efficiency for each project is analyzed to determine the remaining error distribution [20]. Finally, the formal training of code review leaders takes place.

2.3. Iterations in a Software Process

After a process is introduced, it is refined, standardized, implemented, managed, improved, scrutinized and then the users are provided with training. The software process's evolution scale defines a way for the natural evolvement of the process. It appears that in general, evolution is reactive: it occurs to correct the inadequacies of prior implementation of a software engineering practice by adding new layers of practice to support the original process rather than abandoning the process as a whole. That is the software must be designed to be changeable without compromising the confidence in the properties that were initially verified. This repetitive or iterative process of adding layers to support the preceding process is known as Kaizan [20].

3. PRACTICES

Basic practices play a key role in the development processes aiding in viewing the external perspective of an object, which serves as a documentation for the users.

A broad array of concepts, principles, methods, and tools that must be considered as software is planned and developed is known as practice. It represents the details and technical considerations - that underlie the software process - the things, which are needed to actually build high-quality computer software.

Practices differ as one moves from one environment to the next, and they also change as a situation evolves.

3.1. Software Engineering Practice

We can say that a software engineering practice consists of a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis, which transforms a haphazard unfocused approach into something that is more organized, more effective, and more likely to achieve success.

3.2. Software Development Practice

Development is the process of transforming one's ideas into products. Engineers adopt a systematic and organized approach to their work. As one learns software engineering, one should be exposed to many specific practices (or techniques) for developing software. By software development practice we refer to a requirement employed to prescribe a disciplined, uniform approach to the software development process [11, 86], in other words, a well-defined activity that contributes toward the satisfaction of the project goals; generally the output of one practice becomes the input of another practice [11].

3.3 Best Practice

Best practice is a benchmark method or technique of proven procedures that represents the most efficient or prudent course of action that has shown surpassing results juxtaposed to those achieved by other means and therefore can be adapted for some other situation to yield outstanding results.

A best practice is:

"a process, technique, or innovative use of technology, equipment or resources that have a proven record of success in providing significant improvement in cost, schedule, quality, performance, safety, environment, or other measurable factors, which impact an organization [12]."

Or it may be defined according to Williams as:

"a software development practice that, through experience and research, has proven to reliably lead to a desired result and is considered to be prudent and advisable to do in a variety of contexts [11]."

Over time, we accumulate information on whether new practices are good or not. This information might be just stories of people succeeding with the practice, generally called anecdotal or qualitative evidence. Ideally, someone has done a controlled experiment that shows that a new practice is better than some other practice [11].

3.4. Types of practices

The practices need to be tailored and chosen according to the company's need as well as according to individual needs. These may be classified as – *Rudimental, Substrata, Piecemeal practices.*

3.4.1. Rudimental/Radical Practices

They are the training wheels you need to get started and when you take them off, it is evident that you know how to ride. However, remember, that you take them off does not mean you forget how to ride. This is an important difference, which all too often is forgotten in the software. “Yeah we used to write functional specification but we don’t do that anymore,” means you forget to ride, not that you did not need that step anymore [22]. These practices are desiderata, analysis, and feasibility which can be described as follows:

- ***Desiderata***

It is considered necessary and highly desirable in a software development process. This is the requirements phase where the software, hardware, and functional specifications are delineated according to the user needs. Further, the project purpose and scope is also defined. The failure of many software projects can be directly linked to the lack of complete requirement specifications [21].

Boehm addressed the attributes of a good software requirements specification. They included the following: complete, clear, correct, understandable, consistent, concise, and feasible [22].

- ***Analysis***

The technique of breaking down a complex topic into smaller parts to gain a better insight into the subject is known as analysis. Analysis of previous works and projects is done and what modules need to be included is discussed upon. Vitharana and Zahedi (1997) recognized the importance of software requirement analysis in building quality software systems [23].

Just as there are software tools available to assist in the basic building of software code, there are tools that monitor how software is behaving as it runs. These software analysis tools offer visibility into the execution history of an application [85] and allow achieving a higher quality and performance.

Four basic types of analysis tools are:

Code Coverage tool: It measures the amount of the software that has been executed that is the amount of blocks/lines of the code that has been executed. It also generates summary report at the end.

Instruction Trace tool: It is utilized to determine the distribution of instructions and to create a record of exactly what happens as the code is executed.

Memory Analysis tool: It analyzes the allocation and de-allocation of memory locations and identifies possible errors.
Performance Analysis tool: It identifies performance bottlenecks of the application and allows for fine-tuning for achieving higher performance.

In future analyses will be model-driven, namely centred on abstract models of behaviour; modular and incremental, to enable analysis of components, and of systems before completion;

and focussed and partial, rather than uniform, paying closer attention to properties that matter most and to the parts of the software that affect those properties [84].

- ***Feasibility***

A feasibility study is an important phase in the development of business-related services. The need for evaluation is great, especially in large high-risk information service development projects. A feasibility study focuses on the study of the challenges, technical problems and solution models of information service realization, analyses the potential solutions to the problems against the requirements, evaluates their ability to meet the goals and describe and rationalizes the recommended solution [25].

Feasibility literally means whether some idea will work or not. It knows beforehand whether there exists a sizeable market for the proposed product/service, what would be the investment requirements and where to get the funding from, whether and wherefrom the necessary technical know-how to convert the idea into a tangible product may be available, and so on. In other words, feasibility study involves an examination of the operations, financial, HR and marketing aspects of a business before the venture comes into existence [24].

Types of feasibility are:

Technical Feasibility: The term technical feasibility establishes that the product or service can run in the desired way. Technical feasibility means “achievable.” This has to be proved without building the system. The proof is defining a comprehensive number of technical options that are possible within known and demanded resources and requirements. These options should cover all technical sub-areas [25].

Economic Feasibility : This study evaluates the cost of the software development against the cost benefits from the developed system that is there must be scope for profit after the successful completion of the project.

Operational Feasibility: Operational feasibility study tests the working scope of the software to be developed. Higher the operational feasibility, higher is the usability of the system.
Legal Feasibility: It states that the software developed should not violate any privacy act and is within the bounds of the laws and legislations.

3.4.2. Substrata Practices

Substrata practices are the foundational practices for software development process. The foundational practices are the rock in the soil that protects your efforts against harshness of the nature, be it a redesign of your architecture or enhancements to sustain unforeseen growth. They need to be put down thoughtfully and will make the difference in the long haul [26]. These practices are cyanotype, prototyping, and coding which can be described as follows:

- ***Cyanotype and prototyping***

We see more and more companies coming to the realization that modernizing how they do requirements is essential to improve their project’s predictability and success rates. The English scientist and astronomer Sir John Herschel discovered the cyanotype procedure in 1842. This process was one of the first non-silver technologies used to create photographic images. Originated in the 1840's, it was adopted as a copying technique, which became to be known by the term "blueprint," with its blue background reproductions of large engineering, architectural and mechanical drawings.

In, Britain the cyanotype has suffered an almost total aesthetic boycott by photographic artists, connoisseurs, and curators until the last decade or two. In contrast, one can point out to huge archives of cyanotypes where the utility of the process was the paramount consideration. The commercial success of the cyanotype process was owed, not to its pictorial use, but its reprographic applications. These have endowed our language with a new word: 'blueprint', a word that has now taken on an expanded and more abstract meaning. The era of the blueprint as a copying process was heralded by the manufacture of the cheap, sensitized paper in huge quantities. By the turn of the century, its use for copying engineering and architectural plans had become universal in drawing offices [27].

A prototype can be defined as a rudimentary sample, model, exemplar or archetype built to test so that the design can be changed if necessary before the product is manufactured commercially or can be said to be a concept or process or to act as a thing to be replicated or learned from.

- ***Coding***

The coding phase comes under the "most important" category of phases. A code to be effective should consist of the 3 C's, which are clear, consistent, and comprehensible.

Clear: Clear means that the code should be plain and readily apparent to the mind.
Consistent: Being consistent means that the code should be in agreement with itself or with something else and should possess harmony among its parts.
Comprehensible: A code is comprehensible when it is capable of being understood or interpreted.
In addition to all this, a code should be lucid and precise. Therefore, a code should be written with utmost care and concern.

3.4.3. Piecemeal Practices

These practices are the more specialized ones, which provide specific advantages in special conditions. These are the right angle drills – when you need it, there's nothing else that can get between narrow studs and drill a hole perfectly square. At the same time, if there was just one drill you were going to buy, it may not be your first choice [26]. These practices are testing coverage, desegregation, review and scrutiny, tutelage and support which can be described as follows:

- ***Testing Coverage***

The whole program is covered with test cases which are then evaluated based on some fixed criteria which are specified in the requirements. Thus, test cases are used to measure the compliance of the outcome with the desired specifications. Test coverage is of two types: First, done by the professionals who know everything from soup to nuts about the software, and second, done by third persons who do not know about the validity of the functions and code of the software.

- ***Desegregation***

To develop a system various components or parts are manufactured. For the working of the system, these parts need to be integrated so that the system functions coherently. Thus, desegregation is a process of integrating previously segregated parts or components of the system by forming links between them, so that they may function harmoniously and act as an interrelated whole.

- ***Review and Scrutiny***

Review and Scrutiny are to ensure that the product to be delivered is free from defects and errors. This examination done before the final release of the product is known as to review and scrutinize the end product. The removal of errors is necessary so that they do not become defects on delivery to the customers. Once an error becomes a defect, it is known as a software-manufacturing defect. When these manufacturing defects are to be removed, the total cost is incurred on the company manufacturing the software. Therefore, it is necessary to review and scrutinize the software before its delivery to the customers so that no further costs are incurred in travel and transportation and in defect removal of the software.

The Review Process: The value of software review as a mechanism for software quality improvement has been demonstrated repeatedly for over twenty years. Beginning with the landmark work of Michael Fagan at IBM in 1976 structured review mechanisms such as inspection have been shown repeatedly to be an extremely effective means to find work product defects early in the software development process [72].

As the benefits of such a structured review process became more visible, researchers and practitioners began to devise variations on Fagan's original method. For example, Tom Gilb developed a comprehensive inspection method with precisely defined phases, metrics, and suggested process rates for optimum defect removal effectiveness [72, 73].

A review might be either an inspection or a walk-through, without regard to the distinctions made in the software engineering literature. Nearly everyone agrees that reviews work, and nearly everyone uses them, but the way reviews are conducted differ greatly. Most agree that software projects can be routinely completed within time and budget constraints that only a few years ago could be managed only by luck and sweat. Reviews were instituted first for code, and then extended to design. Extensions to requirements and test-case design are not universal, and some feel that the technique may have been pushed beyond its usefulness. Managers would like to extend the review process, while the technical people are more inclined to limit it to the best understood phases of development. Two aspects of reviews must be separated: managerial control and technical utility. Managers must be concerned with both aspects, but technical success cannot be assured by insisting that certain forms be completed [6].

- ***Tutelage and Support***

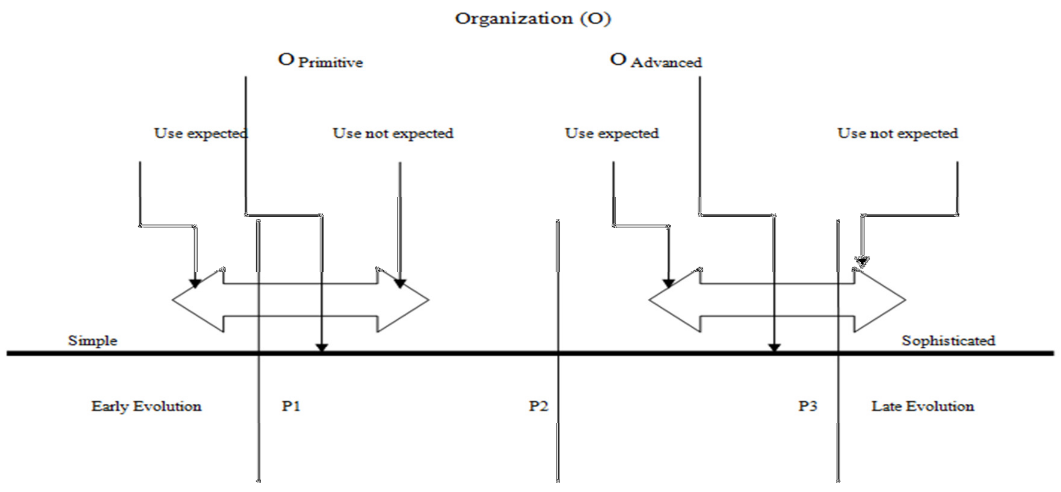
Tutelage is a fundamentally collaborative process, enabling a learner to acquire new concepts and skills from examples or from more experienced people [28]. For example: just as a parachute-glider benefits from the tutelage and support of more experienced parachute- gliders or by some learning examples, similarly end- users can learn the product by the help or tutelage and support provided along with the product and can get assistance as and when required.

3.5. Rasch Analysis

examination of initial data for the possibility of a single latent variable along which software development practices can be calibrated and software development organizations can be measured. When we conceptualize mapping both software practices and organizations on the same continuum, it becomes immediately apparent which software engineering practice one would expect to be used and which would be not expected to be used by an organization at any particular point of evolution [20]. It is important for an organization to know which software engineering practices should be used and which should not be used at a particular point of evolution. Rasch analysis includes the.

3.5.1. Rasch Analysis Model

The Rasch Model specifies within stochastic certainty that the probability of any particular organization using a software engineering practice is a simple function of the sophistication of the organization and the developmental level of the software engineering practice. Specifically, a highly developed organization would have a high probability of using simple software engineering practices [20]. This can be explained with the help of the example:
Figure 3. shows three hypothetical organizations, one primitive and one advanced organization. Conceptually, the primitive organization would be expected to be using software practice p1 and not using practices p2 and p3. Similarly, the advanced organization would be expected to have implemented practices p1 and p2 but not yet p3 [20]. Thus, practices can be calibrated from early to late and the organization should be able to locate its position along the continuum such that it determines which practices are expected to have been implemented and which practices are not expected to have been implemented.



Software Engineering Practices (P)

Figure 2. A hypothetical example of organizations and software engineering practices [20]

The Rasch Model (Wright and Stone, 1979) tests whether this conceptualization describes in a probabilistic sense the actual simultaneous positioning of both organizations and software engineering practices. The expected probability of an organization using a practice is given by Eq. (1).

$$P\ (Xij = 1|\ pi, j) = \frac{e^{(pi - oj)}}{1 + e^{(pi - oj)}}$$

Where p_i is the ith software engineering practice; o_j the jth organization; X_{ij} the response of organization i to practice j; 1 represents used; and 0 represents not used [20].

Thus we can conclude from the model that a primitive organization is expected to use simple practices but not advanced ones while, an advanced organization is expected to use simple as well as advanced practices.

4. SOFTWARE PROCESS

As software is permeating and the quality problem also increasing exponentially, it is necessary to address the problem to ensure that the software-development process is completed within the defined time, is cost-controlled, and excels in quality.

A process is defined as, "a *sequence of steps performed for a given purpose*" according to IEEE and is defined as, "*the organization of: people, automated support, procedures, and standards into work activities designed to produce a specific end result*," according to Software Engineering Institute (SEI) [38]. Process, in a general sense, is composed of three interrelated and interacting ingredients: methods, technologies, and organizations. Methods embody the wisdom of theory and experiences; technologies provide automation of various parts of the process and organizations bound, support or hinder effective processes [13].

After having a look into the terms: software and process, we can now say that a software process deals with the methods and technologies used to assess, support, and improve software-development activities [14]. It may be defined as a "*framework consisting of a set of minimally defined activities or steps aimed at the development of a software"* or " *the set of activities, methods and transformations that people use to develop and maintain software and the associated products, for example: product plans, design documents, codes, test cases and user manuals*," according to SEI [38]. The process adopted depends on the software to be built. One process might be appropriate for creating software for an automobile, while an entirely different process would be required for the creation of a gaming platform.

Process concepts are being applied to software with -increasing success [40-42] but the rate of application of these concepts is limited both by the relatively primitive state of knowledge in this new field and by the lack of a common and precise basis for technical communication [40].
In 2004, Carod, Martin and Aranda said that the present software standards or models are process centric [97]. Process centric means that the process sits at the centre and is the core of organizations, people, technology and management. This is depicted in figure 1:

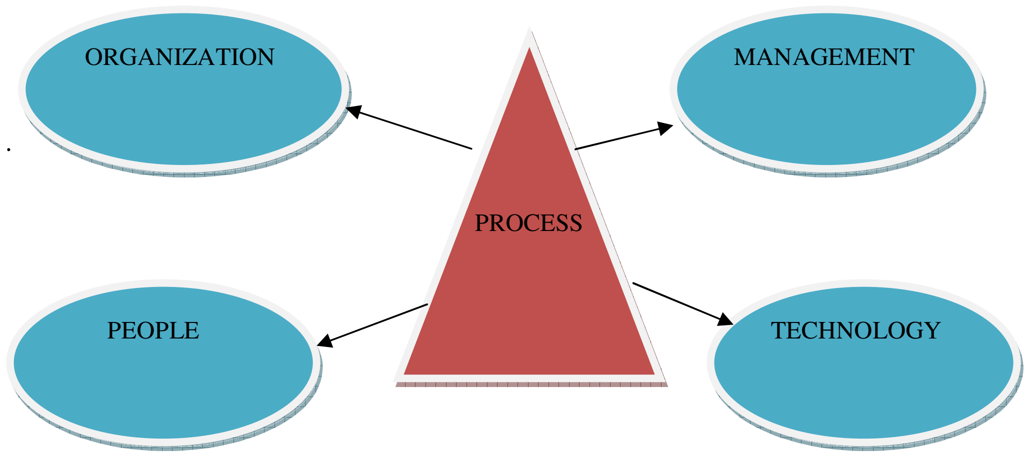


Figure 3. Central role of Process

The field has grown up during the 80s to address the increasing complexity and criticality of software-development activities [14]. It can be viewed in much the same way as software. It has many of the same artifacts and requires quite similar disciplines and methods [39, 40].

Modern software generation is not entirely depended up on process. Even if unbelievable improvements are brought into process, there will be only marginal differences in the form, fit and function of product and the quality of product because today's process based production cannot account the resources and skills aptly [93]. In reality, these days the process itself is getting integrated with software generation intelligence, which is currently, getting transferred to machines [95] which has remained unnoticed [93].

According to Zhong, Madhavji and El Emam (2000), to resolve many issues of the current software industry, there is a need to shift the process centric software generation approach to process and resource centric software generation approach.

Software processes must be evaluated. What constitutes a good process and how can one tell if a particular process fits a specific user need? The first basic requirement is that the properties of a specific process should fit the needs of the project using them [40]. Software process assessments have been widely used for several years and now an increasing number of organizations are conducting such assessments as a business. The software process assessment models are based on the software maturity. Software process maturity implies "the productivity and quality resulting from an organization's software process can be improved over time through consistent gains in the discipline by using the software process [38]." Capability Maturity Model (CMM), Capability Maturity Model Integration (CMMI), People Capability Maturity Model (P-CMM) are some software process assessment models which are discussed as follows:

Capability Maturity Model (CMM): The Software Engineering Institute developed the Capability Maturity Model for Software. This model describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes [43-45]. The CMM is organized into five maturity levels. A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement [45].

Capability Maturity Model Integration (CMMI): The purpose of CMM Integration is to provide guidance for improving your organization's processes and your ability to manage the development, acquisition, and maintenance of products or services. There are multiple CMMI models available, as generated from the CMMI Framework. Consequently, one needs to be prepared to decide which CMMI model best fits one's organization's process-improvement needs [46].

In 2007, it was realized that the present CMMI Model for the software industry is mainly capable of handling Process Management, but the problem persists in their fusion with the models of other aspects [93]. Therefore, an exhaustive rework on the available model is required to make it comply with the other individual models [93, 96, 98-100].

People Capability Maturity Model (P-CMM): This model is a tool to help address the critical issues in the organization. The People CMM helps organizations characterize the maturity of their workforce practices, establish a program of continuous workforce development, set priorities for improvement actions, integrate workforce development with process improvement, and establish a culture of excellence [47].

4.1 Software Development Process

It may be defined as*, "A software development process is the process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes installing and checking out the software for operational use [11,86]."*

4.2. Software Process Management

Software Process management involves the "*planning, monitoring, and control of the process, and events that occur as software evolves from a rudimentary concept to an operational implementation*." Its objective is institutionalizing the activities of software-development with process method to solve the basic problem [37].

There are various problems associated with software process management. Problems like, ad-hoc style of management and technique; non-establishment of a formal process management system [37]; lack of methodology, technology and tools. To address these problems a solution needs to be provided, which includes pervading the software process principle, theory and concept; availability of advanced tools and practices; and to work towards the process improvement. The solution provided must inculcate some features into it.

- **Measurement dependent**

Measurement is one of the important components of software processes. It can be used to identify the challenges faced during the development process and provide a remedy for the same; evaluate the quality of the process; evaluate and ameliorate the capability maturity of the process.
If there were no measurement of the activities, the scenario would have produced lumps of software, with no fulfilled objectives [94].

- **Continual Software Process Improvement**

For the last decade, software process improvement has been a primary approach to improving software quality [55, 61]. Software process improvement refers to a defined framework of software procedures that define the steps and methods of a software process, define measures to assess and benchmark the process, and implement the defined procedures while looking for continuous improvement opportunities. Juran [56, 61], [57, 61], Deming [58, 61] and Crosby [59, 61] have long advocated process improvement as a means to improve quality in product development, manufacturing and services. Humphrey's early research in software engineering highlighted the need for process improvement in the software industry [60, 61]. He drew on principles from manufacturing in developing quality oriented process guidelines for software development, which evolved into the Capability Maturity Model [61, 62]. As the adoption and acceptance of the CMM model have spread, the research literature has become populated with numerous studies establishing a positive relationship between software process improvement and software quality [61-64] and the relationships among process, quality and cost [61, 67, 68-71].

5. IMPROVING THE SOFTWARE DEVELOPMENT PROCESS

Some very small changes can improve productivity in many installations. While there is no empirical evidence that will permit us to forecast gains, there is a general consensus in the software community (like that for the use of high-level languages) that supports these ideas [6]:

- **Iterative Enhancement**

Several techniques have been suggested as aids for producing reliable software that can be easily updated to meet changing needs [76, 77]. These include the use of a top-down modular design, a careful design before coding, modular well-structured components, and a minimal number of implementers. It is generally agreed that the basic guideline is the use of a top-down modular approach using "stepwise refinement" [78] known as iterative enhancement. This technique is a practical approach to software-development that begins with a simple initial implementation of a properly chosen subproject, which is followed by the gradual enhancement of successive implementations in order to build the full implementation [75].

- **Evaluating Methods and Tools**

A software tool is a system, which assists in some phase of the software development process. Therefore, it becomes necessary to constantly evaluate the tools and method in order to work towards software process improvement. In fact, a separate organization with this charter should be established [6] which totally focuses on the tool evaluation, their findings and according to these findings provide a better tool or methodology for the software-development.

- **Improving the Review Process**

Review is extremely labor-intensive. Typical procedures for FTR involve individual study of hard-copy designs or source listings and hand-generated annotations, followed by a group meeting where the documents are paraphrased line by line, issues are individually raised, discussed, and recorded by hand, leading eventually to rework assignments and resulting changes [74]. Thus, the manual labor should be reduced so that the reviewers can spend more time and effort in finding out the logical errors rather than spending time on the formatting and syntactic errors. This can be possible by the use of tools and methodologies, which can be used to resolve the formatting anomalies. These tools help strengthen the software review process.

CONCLUSION

The computer-based market is worth hundreds of billions of USD, which affects the people in some or the other way. Thus, it is necessary to have qualified software professionals. These software professionals have the responsibility to adopt the best practices in the industry to increase the success rate of their projects. This itself is a great responsibility in hand for the software people. It becomes important for them to search various practices and adopt the most suitable for the organization. Adopting the practice needs the understanding of the calibration of the practices with the organization. Further, software engineering techniques often pay heed only on the initial software-development and not on the continual evolution of the software. Software is flexible; it is continually evolving primarily due to changing or evolving needs, advancement of technology, changes in the operational environment and due to discovery of latent errors. This evolution enables the software to retain its previous characteristics and attain new characteristics, which work towards the melioration of the software. Only evolving the software does not help, advanced tools and methodologies are also needed and must be constantly evaluated for their efficiency and effectiveness check.

ACKNOWLEDGMENT

This work was supported by Jaipur Engineering College and Research Centre (JECRC). We are thankful to Mr. Arpit Agarwal (Director, JECRC) for valuable suggestions, kind support and encouragement. Further, also want to convey thanks to Dr. Ekta Menghani (Biotechnology Department, MGIAS,) Associate Professor Manish Tiwari (E&C Department, JECRC) for their time-to-time suggestions and technical support.

REFERENCES

[1] Shihab A. Hameed, Khalid Al and Khateeb, Zubayda Mutaz, "*Software Engineer Islamic Ethics an Interactive Web-Based Model* ."

[2] Boehm, B.W. "*Improving Software Productivity*," IEEE Computer (20. ^), September 1987, pp.43-57.

[3] Tridas Mukhopadhyay , Steven S. Vicinanza and Michael J. Prietula , "*Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation,"* Carnegie Mellon University,U.S.A.

[4] L. Osterweil "*Software Processes are Software too*," in Proceedings of the 9th. International Conference on Software Engineering, ACM Press, New York, N.Y., pp. 2-13, 1987.

[5]Matteo Gaeta, Consorzio CRMPA – Centro di Ricerca in Matematica Pura ed Applicata, Pierluigi Ritrovato, Dipartimento di Ingegneria dell'Informazione e Matematica Applicata – Università di Salerno Via ponte don Melillo – 84084 Fisciano – Italy, "*Generalised Environment for Process Management in Cooperative Software Engineering."*

[6] Marvin V. Zelkowitz, Raymond T. Yeh, Richard G. Hamlet, John D. Gannon, and Victor R. Basili, "*Software Engineering Practices in the US and Japan*," University of Maryland.

[7] Fritz Bauer, "*40 Years of Software Engineering, A Look Back*", Keynote Speech at ICSE 2008, Presented by Manfred Broy.

[8] Binghui Helen Wu, " *On Software Engineering and Software Methodologies, A Software Developer's Perspective*."

[9] Mary Shaw, " *Prospects for an Engineering Discipline of Software,"* IEEE Software Dec. 1990.

[10] Barry Boehm, "*A View of 20th and 21st Century Software Engineering*, " Keynote speech at ICSE 2006.

[11] Dr. Laurie Williams, *"A (partial) Introduction to Software Engineering Practices and Methods."*

[12] Javelin Technologies, *"Best Practice Definition*," Oakville, Ontario, Canada, 2002.

[13] Dewayne E. Perry, Professor and Motorola Regents Chair in Software Engineering Electrical and Computer Engineering, The University of Texas at Austin,"*Dimensions of Software Evolution*."

[14] Alfonso Fuggetta, Politecnico di Milano, "*Software Process: A Roadmap*."

[15] Michele Lanza, Faculty of Informatics University of Lugano, Switzerland, "*Software Evolution*."

[16] *Software and System Safety Research Group: A White Paper* by Nancy Leveson Aeronautics and Astronautics Massachusetts Institute of Technology.

[17] E.Karch, "*Lehman's Laws of Software Evolution and the Staged – Model*."

[18] Lehman MM, Ramil JF,Wernick PD and Turski WM.Metrics , *"laws of software evolution— the nineties view*," Proceedings of the 4th International Software Metrics Symposium (Metrics '97), IEEE Computer Society Press: Los Alamitos CA, 1997; 20.

[19] M.Ahmed nacer, Computer System and Applications ACSIEEE International Conference, 2001, "*Towards a new approach on software process evolution*."

[20] David E.Drehmer, Sasa M.Dekleva, 2000, *A note on the evolution of software engineering practices."*

[21] Connolly, T., Begg, C., & Strachan, A. (1999*), " Database Systems: A Practical Approach to Design, Implementation and Management (Second Ed.)*," Reading MA: Addison – Wesley.

[22] Boehm, B. (1984), "*Verifying and validating software requirements and design specifications*," IEEE Software, 1(1), 75-88.

[23] Vitharana, P., & Zahedi, F. (1997), "*Group Decision Support for Software Requirements Analysis. Association for Information Systems. "*

[24] Dr. Anand Saxena, Seema Sodhi, "Lesson – 5 *Feasibility Analysis, Project Report and Business Plan."*

[25] Teppo Kivento*, "Technical feasibility."*

[26] Ram Chillarege, Center for Software Engineering, IBM Research, "*Software Testing Best Practices*."

[27] Mike Ware, "*Cyanotype, The history, science and art of photographic printing in Prussian blue*."

[28] Andrea Lockerd and Cynthia Breazeal, "*Tutelage and Socially Guided Robot Learning*."

[29] M.Shaw , "*The coming age of software architecture research*," Proceedings of the International Conference on Software Engineering May (2001).

[30] W.F.Tichy, P.Lukowicz, L.Prechelt and E.Heinz, "*Experimental evaluation in computer science: a quantitative study*," Journal of Systems and Software Jan (1995).

[31] M.V.Zelkowitz and D.Wallace, "*Experimental validation in software engineering, Information and Software Technology,"* 39 (1997) 735- 743.

[32] N.Fenton, S.L.Pfleeger and R.L.Glass*, "Science and Substance: a challenge to software engineers*," IEEE Software July (1994).

[33] C.Potts, "*Software engineering research revisited*," IEEE Software May (1993).

[34] R.L.Glass, "*The software-research crisis*," IEEE Software Nov (1994).

[35] R.L.Glass, I.Vessey and V.Ramesh*, "Research in software engineering: an analysis of the literature*," Elsevier, Information and Software Technology 44 (2002) 491- 506.

[36] F.Land, Leo*, "the first business computer: a personal experience*, in: R.L.Glass (Ed.)., *In the Beginning: Recollections of Software Pioneers*," IEEE Computer Society Press, New York, 1998.

[37] Qing Wang , *"Software Process Management: Practices in China*," Institute of Software Chinese Academy of Sciences.

[38] Dr. Sami Zahran*,"Software Process Improvement using Capability Maturity Model "CMM."*

[39] Osterweil, L.J., "*Software Processes are Software Too*," Proceedings of 9th International Conference

on Software Engineering (ICSE9), IEEE Computer Society Press, April 1987.

[40] A Peter H. Feller, Watts S. Humphrey, "*Software Process Development and Enactment: Concepts and Definitions*," September 1992.

[41] Humphrey, W.S., Snyder, T,.F and Willis, R.R., "*Software Process Improvement at Hughes Aircraft*," IEEE Software, July 1991, pp. 11-23.

[42] Kolkhorst, B.G. and Macina, A.J. "*Developing Error-Free Software*," Proceedings of Computer Assurance COMPASS '88, NIST, IEEE, July 1988.

[43] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "*Capability Maturity Model for Software, Version 1.1,"* (CMU/SEI-93-TR-24, ADA 263403), Pittsburgh, PA: Software Engineering Institute, February 1993.

[44] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush. "*Key Practices of the Capability Maturity Model,Version 1.1,"* (CMU/SEI-93-TR-25, ADA 263432), Pittsburgh, PA: Software Engineering Institute, February 1993.

[45] Mark C. Paulk, "*A Comparison of ISO 9001 and the Capability Maturity Model for Software,"* Carnegie Mellon University, Software Engineering Institute, 7-1-1994.

[46] "*Capability Maturity Model® Integration (CMMI^{SM}), Version 1.1"*, (CMU/SEI-2002-TR-028, ESC-TR-2002-028) CMMI^{SM} for Software Engineering (CMMI-SW, V1.1), Carnegie Mellon, Software Engineering Institute, August 2002.

[47] Bill Curtis, Bill Hefley and Sally Miller, "*People Capability Maturity Model (P-CMM) Version 2.0, Second Edition*," Carnie – Mellon University, Software Engineering Institute.

[48] Belady L and Lehman M. , "*A model of large program development*," IBM Systems Journal 1976; 15(3):225–252.

[49] Evelyn J. Barry, Chris F. Kemerer, and Sandra A. Slaughter, "*How software process automation affects software evolution: a longitudinal empirical analysis*," Journal Of Software Maintenance And Evolution: Research And Practice, J. Softw. Maint. Evol.: Res. Pract. 2007; 19:1–31, Published online in Wiley InterScience.

[50] Lehman MM, Ramil JF, "*Software evolution—background, theory, practice,"* Information Processing Letters 2003; 88(1–2):33–44.

[51] Belady LA, Lehman MM, *"Program Evolution: Processes of Software Change*," Academic Press: London, 1985; 538 pp.

[52] Lehman MM, "*On understanding laws, evolution and conservation in the large program life Cycle*," Journal of Systems and Software 1980; 1(1):213–221.

[53] T. Mens and K.Mens, "*Assessing the evolvability of software architectures*," In Proceedings of the ECOOP'98, 1998.

[54] Robert Brcina, Stephan Bode and Matthias Riebisch, "*Optimisation Process for Maintaining Evolvability during Software Evolution*," Technical University of Ilmenau, Germany.

[55] T. Dyba, "*An Empirical Investigation of the Key Factors for Success in Software Process Improvement*," IEEE Trans. Soft-ware Eng., vol. 31, no. 5, pp. 410-424, 2005.

[56] J.M. Juran, "*A Note on Economics of Quality*," Industrial Quality Control, pp. 20-23, 1959.

[57] J.M. Juran, "*Juran on Quality by Design: The New Steps for Planning Quality into Goods and Services*," Free Press, 1992.

[58] W.E. Deming, "*Out of Crisis*," MIT Center for Advanced Engineer-ing Study, 1992.

[59] P.B. Crosby, "*Quality is Free*," McGraw-Hill, 1979.

[60] W.S. Humphrey, "*Characterizing the Software Process: A Ma-turity Framework*," IEEE Software, vol. 5, no. 3, pp. 73-79, 1988.

[61] Donald E., Kemerer , Chris F., Member, IEEE Computer Society and Slaughter and Sandra A, " *Software Process Improvement Reduce the Severity of Defects?*," A Longitudinal Field Study Harter, IEEE Transactions On Software Engineering.

[62] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "*Capabili-ty Maturity Model, Version 1.1*," IEEE Software, vol. 10, no. 4, pp, 18-27, 1993.

[63] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes and M. Paulk, "*Software quality and the Capability Maturity Model*," Comm. of ACM, vol. 40, no. 6, 1997.

[64] G. Li and S. Rajagopalan, "*Process Improvement, Quality and Learning Effects*," Management Science, vol. 44, pp. 1517-1532, 1998.

[65] M.S. Krishnan and M.I. Kellner, "*Measuring Process Consisten-cy: Implications for Reducing*

Software Defects," Management Science, vol. 25, pp. 800-815, 1999.

[66] N. Ramasubbu, S. Mithas, M. S. Krishnan, and C. F. Kemerer, "*Work Dispersion, Process-Based Learning and Offshore Soft-ware Development Performance*," MIS Quarterly, v. 32, n. 2, pp. 437-458, June 2008.

[67] M.S. Krishnan, C.H. Kriebel, S. Kekre and T. Mukhopadhyay, "*An Empirical Analysis of Productivity and Quality in Software Products*," Management Science, vol. 46, no. 6, 2000.

[68] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "*Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development*," Management Science, vol. 46, pp. 451-466, 2000.

[69] H. Wohlwend and S. Rosenbaum, "*Schlumberger's Software Improvement Program*," IEEE Trans. Software Eng., vol. 20, no. 11, pp. 833-839, 1994.

[70] M. Diaz and J. Sligo, "*How Software Process Improvement Helped Motorola*," IEEE Software, vol. 14, no. 5, pp. 75-81, 1997.

[71] M. Agrawal and K. Chari, "*Software Effort, Quality and Cycle.*"

[72] Adam A.Porter and Philip M.Johnson, "*Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies,*" Feb 1997.

[73] Tom Gilb, "Dorothy Graham, " *Software Inspection*," Addison-Wesley, 1993.

[74] Philip M. Johnson and Danu Tjahjono, "*Improving Software Quality through Computer Supported Collaborative Review*," Proceedings of the Third European Conference on Computer Supported Cooperative Work, Milan, Italy, September 1993.

[75] Victor R.Basili and Albert J.Turner, "*Iterative Enhancement: A Practical technique for Software Development*", IEEE Transactions on Software Engineering, Vol. SE-1, No.4, Dec 1975.

[76] H.D.Mills, "*On the development of large, reliable programs*," Rec.1973 IEEE Symp. Comp. Software Reliability, Apr. 11973, pp. 155-159.

[77] D.L.Parnas," *On the criteria to be used in decomposing systems into modules,*" Commun. Ass. Comput. Mach, Vol 15, pp. 1053-1062, Dec. 1972.

[78] N.Wirth, "*Program development by stepwiswe refinement,*" Commun, Ass.Comput. Mach., Vol. 14, pp. 221-227, Apr. 1971.

[79] S. L. Pfleeger, "*Software Engineering: Theory and Practice*", Upper Saddle River, NJ: Prentice Hall, 1998.

[80] F. P. Brooks, " The Mythical Man-Month, Anniversary Edition": Addison-Wesley Publishing Company, 1995.

[81] W. W. Gibbs, "*Software's Chronic Crisis*," in Scientific American, 1994, pp. 86-95.

[82] Robert L.Glass, "*The Software Research Crisis*," Computing Trends, IEEE Software.

[83] Ian Sommerville , "*Software Engineering*," Seventh Edition. Pearson.,Addison Wesley – Boston, ISBN 0-321-21026-3, 2004.

[84] Daniel Jackson and Martin Rinard, "*Software Analysis: A Roadmap*," Laboratory for Computer Science, Massachusetts Institute of Technology.

[85] "*Techniques and Tools for Software Analysis*," Freescale Semiconductor.

[86] IEEE, "*IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology*," 1990.

[87] Mika Ahokas, Jyrki Kontio, Markus M. Mäkelä, Päivi Pöyry and Aki Lassila, "*Effects of Software Engineering Practices on the Scalability of Firms' Software Development Output.*"

[88] Cusumano, M., *"The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad",* Free Press, New York, 2004.

[89] Messerschmitt, D. and Szyperski, C. , "*Software Ecosystem*", MIT Press, London, 2003.

[90] Mowery, D., "*The International Computer Software Industry: A Comparative Study of Industry Evolution and Structure*", Oxford University Press, New York, 1996.

[91] Brown, S. L., Eisenhardt, and K. M., "*The art of continuous change: Linking complexity theory and time-paced evolution in relentlessly shifting organizations, Administrative Science Quarterly*", Vol. 42(1), pp. 1-34, 1997.

[92]Juan F Ramil and Meir M Lehman, "*Evolution in the Era of Component Based Software.*"

[93]Muthu Ramachandran and Rogério Atem de Carvalho, "Software Engineering and Productivity Technologies", 2010.

[94]Rauterberg, M., and Aeppli, R., "How to measure the behavioural and cognitive complexity in Human-Computer Interaction with Petri nets," IEEE Workshop on Robot and Human

Communication, 1996.
[95]Menkhaus, G., Frei, U., and Wuthrich, J, "*Analysis and Verification of the Interaction Model in Software Design*," 2006.
[96]Hoffmann, H.F., Yedlin, D.K., Mishler, J.W., and Kushner, S., "*CMMI for Outsourcing Guidelines for Software, Systems and IT Acquisition*," Reading, MA: Addison Wesley Professional, 2007.
[97]Carod, Martin and Aranda , *http://www.sei.cmu.edu/cmmi*, 2004.
[98] Siviy, J.M., Penn, M.L., and Robert, W., "*Standard CMMI (R) and Six Sigma: Partners in Process Improvement,"* Reading, MA: Addison Wesley Professional, 2007.
[99] Chrissis, M.B., Konrad, M., and Shrum, S., "*CMMI (R): Guidelines for Process, Integration and Product Improvement (2nd Ed.)"*, Reading, MA: Addison Wesley Professional, 2003.
[100] Ebert, Christ of Akins and Anthony, Book-shelf, *Software* 24(3), 110 -112, 2007.

AUTHORS

Sunil Kumar Jangir

Mr.Sunil Kumar Jangir is an Assistant Professor in the Department of Information Technology of Jaipur Engineering College & Research centre, Jaipur. He is currently pursuing his M.Tech in Software Engineering from Gyan Vihar University, Jaipur. He has 2 years and 6 months of teaching and industrial experience. He has presented 10 research papers in various International and National Conferences. His research interests include Software Engineering, Knowledge Management, Information security, Software Project Management.

Neha Gupta

Ms. Neha Gupta is an Assistant Professor in the Department of Information Technology of Jaipur Engineering College & Research centre, Jaipur. She is currently pursuing her M.Tech. in Software Engineering from Gyan vihar University, Jaipur and has 6 years of teaching experience. Her research interests include Software Engineering, Database Management System.

Shreya Agrawal

Ms. Shreya Agarwal is a Student of Jaipur Engineering College and Research Centre under Rajasthan Technical University and is currently in the final year of Bachelor of Technology with Information Technology as her major. Her main subje cts of interests are Software Engineering, Software Project Management, Database Management System and Data Mining and Warehousing.