ISSN 1512-1232

AN EXPERIMENTAL EXAMINATION OF EFFECTIVE SOFTWARE PRODUCTION PROCESS: A NOVEL STUDY

Dr.S.S.Riaz Ahamed

Principal, Sathak Institute of Technology, Ramanathapuram, Tamilnadu, India.

Email:ssriaz@ieee.org, ssriaz@yahoo.com

ABSTRACT

In this era of globalization which is characterized by intense competition, the software industry moves unrelentingly towards new methods for managing the ever-increasing complexity of software projects. We have constantly witnessed evolutions, revolutions, and recurring themes of success and failure. The technologies, process, and methods have advanced rapidly, software project management and quality assurance remains a people-intensive process.

1 INTRODUCTION

To find an organization's present position, the SEI (Software Engineering Institute) uses an assessment questionnaire and a five-point grading scheme. The grading scheme determines compliance with a model known as the capability maturity model that identifies key set of activities required at various levels or process maturity.

Capability Maturity Model

CMM is not a software process model. Instead, it is a strategy for improving the software process.

(1) CMM assists organizations in providing the infrastructure for a disciplined and mature software process. The CMM strategy is to improve the management of the software process, in the belief that this will lead to improvements in techniques.

(2) **Maturity Levels**:

Maturity Level	Characterization
1. Initial	Ad hoc process
2. Repeatable	Basic project management
3. Defined level	Process definition
4. Managed level	Process measurement
5. Optimizing level	Process control

(3) An organization can assess maturity using a series of questionnaires developed by CMU's SEI (Software Engineering Institute). Thus this approach is good for organization, which must access software procurement.

(4) The software maturity model emphasizes measurement, training and retraining of software personnel, and quality control of the software process. However, it also increases productivity, as a software development organization moves up to higher level of maturity.

The SEI method ascertains global effectiveness and outlines the following five maturity levels.

Level 1: Initial - The initial software process is one in which has a few processes defined intermittently and even with elements which lead to confusion. Success is often attributed to the individual effort.

ISSN 1512-1232

Level 2: Repeatable - The essence of project management process are built in to track cost, schedule and functionality. The essential project discipline is set in right place to ensure successes on projects, which are similar.

Level 3: Defined - Both the management and the engineering activities of a software process are aptly documented, standardized, and integrated into an organization-wide software process.

Level 4: Managed - The software process and the associated products are quantitatively and qualitatively collected, understood and controlled using elaborate measures.

Level 5: Optimizing - The improvement in the process is attained by continuous quantitative feedback from the process and from testing innovative ideas and technologies and it obviously fulfills level 4.

The five levels defined by the SEI are derived because of evaluating responses to the SEI assessment questionnaire that is based on the capability maturity model. The results of the questionnaire are distilled to a single numerical grade that provides an identification of an organization's process maturity. The SEI has associated key process areas with each of the maturity levels. The KPAs describe those software engineering functions (e.g. software project planning, requirements management) that must be present to satisfy good practice at a particular level.

The key performance areas are identified by the following characteristics:

- *Goals*- the overall objectives that the KPA must achieve.
- *Commitments*- requirements that must be met to achieve the set goals.
- *Abilities* - Set of things in the right place for achieving organization's commitments.
- *Activities* - Tasks to achieve KPA.
- *Implementation monitoring* - methods in which activities are controlled and regulated as they are put in the right place.
- *Implementation verifying* - KPAs defined across the maturity model and mapped into different levels of process maturity.

Process maturity level 2

- Software configuration mangement
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight.
- Software project planning
- Requirements management

Process maturity level 3

- Peer reviews
- Intergroup coordination
- Software product engineering
- Integrated Software Management
- Training program
- Organization process definition
- Organization process focus

Process Maturity level 4

- Software quality management
- Quantitative process management

ISSN 1512-1232

Process Maturity level 5

- Process change management
- Technology change management
- Defect prevention

2 SOFTWARE PROCESS MODELS

In order to have a proper understanding of the subject regarding software, an individual is required to understand different models and the significance that it has in the present day situation to design a software. For solving an actual problem in an industrial setting, a software engineer or a team of engineers must incorporate development strategy that encompasses the process, methods, and tools layers and the generic phases. This strategy is often referred to as a process model or a software engineering paradigm.

2.1 Waterfall Model:

This model is considered the simplest process model. This model is also referred to as the linear sequential model or the "classic life cycle". This model states that phases are organized in a linear order. Depending upon the nature of activities and the flow of control among them, there are various variations to this model. In a typical model, a project often begins with the feasibility analysis. On successfully demonstrating the feasibility of the project, the requirement analysis and project planning starts. The design starts after the requirements analysis is complete. When the programming is completed, the code is integrated and testing is done. On successful completion of testing, the system is installed. After this the regular operation and maintenance of the system takes place. It can be observed that from the waterfall model, the sequence of activities performed in a software development project is Feasibility study, Requirement analysis and specification, Design and Specification, coding and module testing, systems integration and testing, delivery and maintenance.

Activities of the Waterfall Model

A Peep into the project outputs in Waterfall Model:

The output of a project employing the waterfall model is not just the final program along with the documentation to use. There are a number of intermediate outputs that must be produced to produce a successful product. The set of documents that should be produced in a project is dependent on how the process is implemented; the following is a set of documents that generally forms the minimum set that should be produced in each project.

- Feasibility study
- Requirement analysis and specification
- Design and Specification
- Coding and module testing
- Systems integration and testing
- Delivery
- Maintenance

It can be inferred that reviews are necessary, especially for the requirements and design phases, because other certification means are certainly not available. Reviews are meetings conducted to undress deficiencies in a product. The review reports are the out of these reviews.

Feasibility study

The Phase includes

ISSN 1512-1232

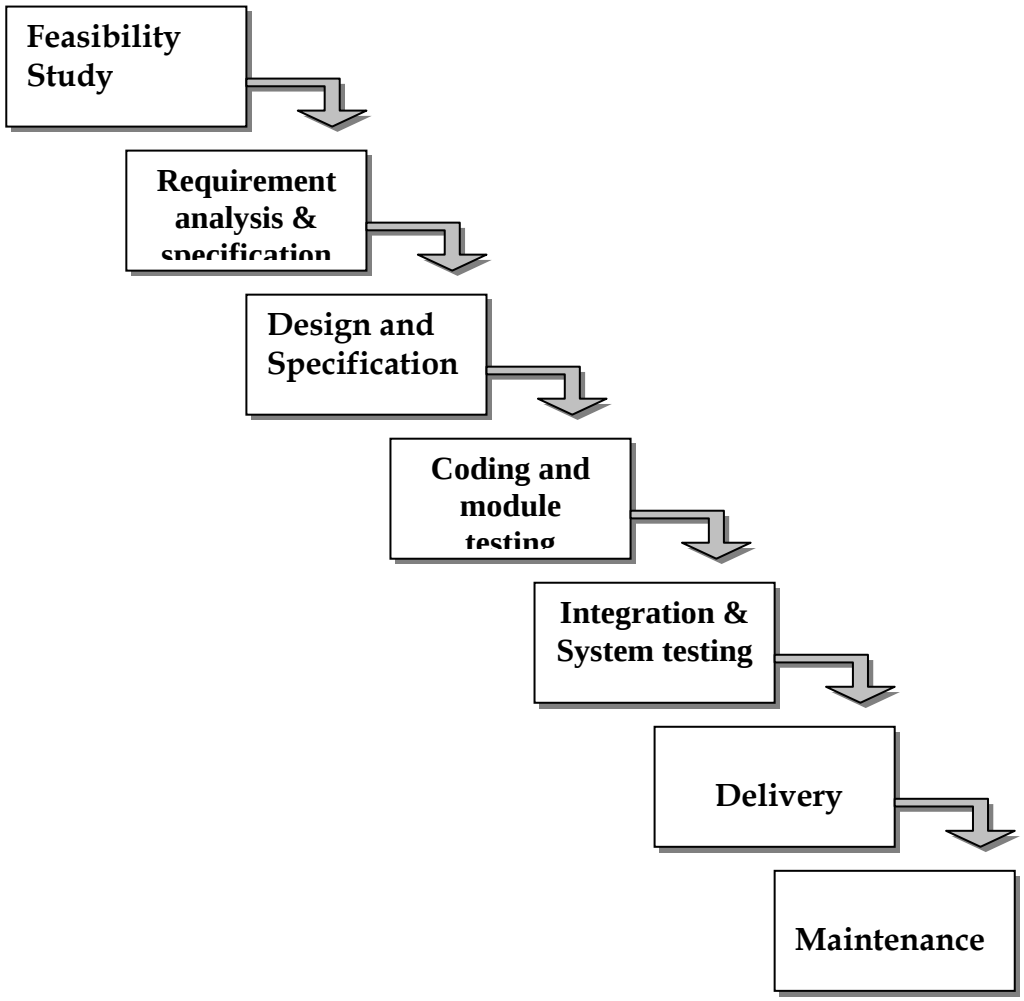
1. Problem definition
2. Solutions on various aspects and their benefits
3. Infrastructure and Cost Estimation of benefited solutions.

Requirement analysis and specification

This phase defines the quality of the project proposed and the required resources. The outcome of this phase is requirements specification document and the purpose of this document is twofold: one for customer satisfaction (looking ahead the proposal whether it meets the requirement) and the other for the sake of Software engineers. The document should be understandable, precise, complete, consistent, unambiguous, and modifiable. The requirements list can be classified as (i) functional requirements (ii) non-functional requirements and (iii) Requirements on the development and maintenance process.

Design and Specification

The phase involves in decomposing the system into modules and a description report "design specification document" is produced. The output helps to retrieve the information of different modules and their respective functions.



Coding and module testing

Coding and module testing is the phase where actually the decomposed modules are computerized using effective programming languages. The phase also involves the major quality control activity – module testing that involves in testing each modules.

ISSN 1512-1232

Systems integration and testing

Integration amounts to assembling the application from the set of components that were developed and tested separately. As a final stage, the development team tests the entire system.

Delivery

The delivery is done in two ways:

(i) The system is tested by selected customers and the process is called as beta testing and
(ii) The product is delivered to customer community.

Maintenance

Maintenance is defined as a set of activities that are performed after the system is delivered to the customer. It generally consists of

- Correcting of any remaining errors (corrective maintenance)
- Changing the application based on the environment (adaptive maintenance)
- Improving, changing or adding features (perfective maintenance).

It must be mentioned that linear ordering of activities has some important consequences. First, to clearly identify the end of a phase and the beginning of the next, some certification mechanisms has to be employed at the end of each phase. The verification and validation means ensures that the output of a phase is consistent with its input (which is the output of the previous phase), and that the output of the phase is consistent with the overall requirements of the system. The need for certification is that each phase must have some defined output that can be evaluated and certified. That is, when the activities of a phase are completed, there should be some product that is produced by that phase. In addition, the goal of a phase is to produce the product. The output of the earlier phase is to produce this product. The outputs of the earlier phases are referred to as work products (or intermediate products) and are generally in the form of documents like the requirements document or design document. For the coding phase, the output is the code. The output of a software project is not just the final program along with the user documentation, but also the requirements document, design document, project plan, test plan, and test results.

2.2 Prototyping Model

One of the main objectives of the prototyping based development process is to get over the first two limitations of the waterfall model. The idea behind prototyping is that instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to understand the requirements. Of course, the development of the prototype undergoes design, coding and testing, but each of these phases is not done very formally and fully. By making use of the prototype, the person gets an actual feel of the system, which ultimately paves the way for better understanding of the requirements of the system. It is indeed accepted that prototyping is an attractive idea for complicated and large system for which there is no manual process or existing system to help determine the requirements. Allowing the client “play” with the prototype provides invaluable and intangible inputs that help determine the requirements for the system. It is an effective method for determining the feasibility of certain approach.

The prototype can serve as the “first system”. Both customers and developers like prototyping paradigm. Users get a feel of the actual system and developers get to build something immediately.

Prototyping can be an effective paradigm for software engineering. The key is to define the rules of the game at the beginning; that is, the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part), and the actual software is engineered with an eye towards quality and maintainability.

ISSN 1512-1232

2.3 The RAD Model

Rapid Application Development is a linear sequential software development process, which emphasizes an extremely short development cycle. The RAD is a "high speed" model in which rapid development is achieved by using a component-based construction approach. Fully functional system can be attained within a very short time in this model provided the requirements are fully understood.

2.4 The Incremental Model

The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. Under this model, the first is often the generation of a core product. Then the supplementary issues are addressed by use and evaluation by creation of an incremental plan. The plan is implemented and the user checks the additional features until the complete product is born.

The incremental process model is iterative in nature. However, unlike prototyping, the incremental model focuses on the delivery of operational product with each increment. Early increments are "stripped down" versions of the final product, but they do provide capability that serves the use and also provide a platform for evaluation by the user.

It is a model whose stages consist of expanding increments of an operation software product, with the direction of evolution being determined by operational experience. Increments may be delivered to the customer as they are developed; this is referred to as evolutionary or incremental delivery. We may summarize the development strategy of an evolutionary process model:

1. Deliver something to the real user.
2. Measure the added value to the user in all critical dimensions.
3. Adjust both the design and the objectives based on observed realities.

This model may be also termed as incremental implementation and delivery model.

2.5 The Spiral Model

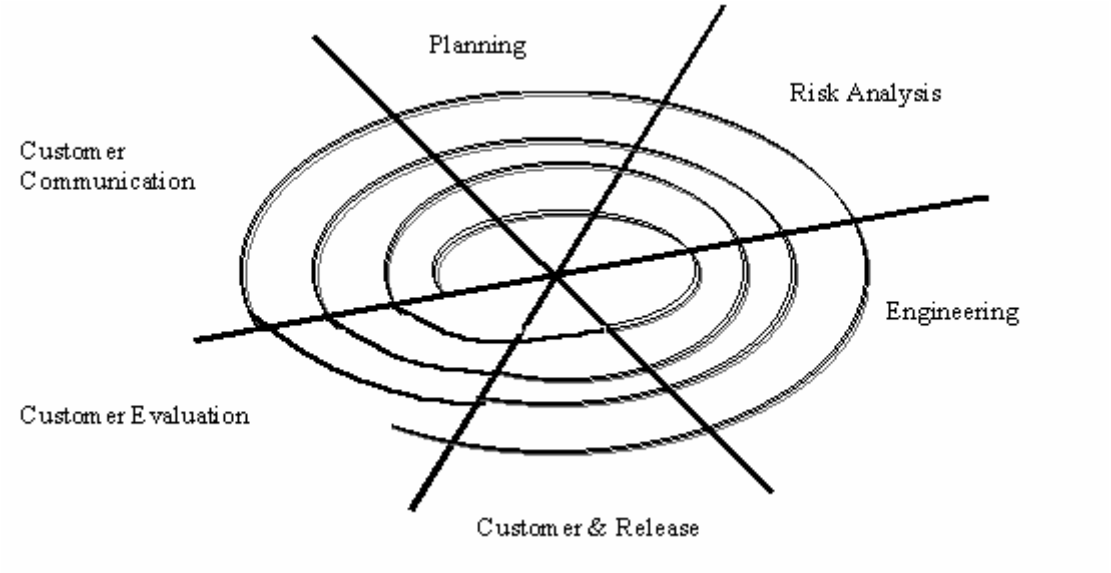
The model is recent in origin. Boehm has proposed this model. True to its name, the activities in this model can be organized like a spiral that has many cycles. The radial dimension represents the cumulative cost incurred in accompanying the steps done so far, and the angular dimension represents the progress made in completing each cycle of the spiral. Every cycle in the spiral begins with the identification of the objectives for the cycle, the different alternatives that are possible for achieving the objectives, and the constraints that exist.

As the model is relatively new, it has the property of encompassing different development strategies apart from development activities. For projects that have a high degree of risk, this model is preferred as it properly takes care of the management and planning activities.

The spiral model is divided into a number of framework activities, also called task regions.

- Customer communication – tasks required to develop effective communication between developer and customer.
- Planning – tasks required to define resources, timelines, and other project related information.
- Risk analysis - tasks required to accomplish both technical and management risks.
- Engineering: - tasks required to build one or more representations of the application.
- Construction and release- tasks required to construct, test, install and provide user support (e.g. documentation and training)
- Customer evaluation – tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.
-

ISSN 1512-1232



2.6 The Component Assembly Model

Object technologies provide the technical framework for a component based process model for software engineering. The object-oriented paradigm emphasizes the creation of lasses that encapsulate both data and the algorithms that are used to manipulate the data. If properly designed and implemented, object oriented classes are reusable across different applications and computer based systems architectures.

Under this model, data and algorithm are packaged into a class. Classes (called components), created in the past are stored in class library. The class library is referenced and the right classes are identified and reused.

Thus, component assembly model leads to software reuse and reusability, which has a number of benefits. Based on studies by QSM Associates, it is found that component assembly leads to a 70% reduction in development cycle time, an 84% reduction in project cost, and a productivity indeed of 26.2 compared to an industry norm of 16.9.

2.7 The Concurrent Development Model

It is a schematic representation of a series of major technical activities, tasks and their associated states. The concurrent process model defines a series of events that will trigger transitions from state to state for each of the engineering activities. Concurrency is achieved in two ways viz

(1) System and component activities occur simultaneously and can be modeling using the state oriented approach using the above;
(2) A typical client server application is implemented with many components, each of which can be designed and realized concurrently. The concurrent process model is more practical and can be applied to all types of software development and provides a more correct and accurate picture of the current state of the project. It defines a network of activities. Each activity on the network exists simultaneously with other activities. Events generated within a given activity or at some other place in the activity network trigger transitions among the states of an activity.

It is a schematic representation of a series of major technical activities, tasks and their associated states. The concurrent process model defines a series of events that will trigger transitions from state to state for each of the engineering activities. Concurrency is achieved in two ways viz

ISSN 1512-1232

(1) System and component activities occur simultaneously and can be modeling using the state oriented approach using the above;
(2) A typical client server application is implemented with many components, each of which can be designed and realized concurrently. The concurrent process model is more practical and can be applied to all types of software development and provides a more correct and accurate picture of the current state of the project. It defines a network of activities. Each activity on the network exists simultaneously with other activities. Events generated within a given activity or at some other place in the activity network trigger transitions among the states of an activity.

3 SOFTWARE LIFE CYCLE

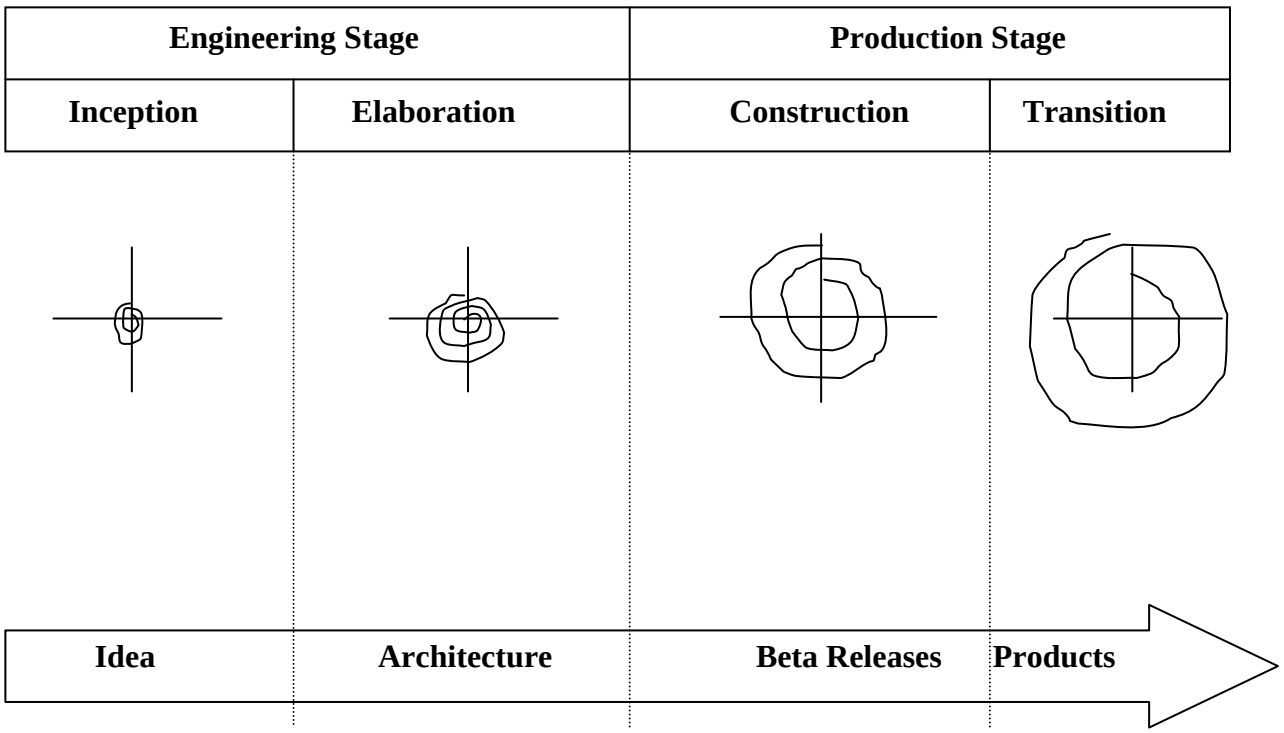
The software life cycle is broken into cycles with each cycle working on a generation of the system. To achieve economies of scale and higher returns on investment, we must move toward a software manufacturing process driven by technological improvements in process automation and component-based development. The life cycle is divided into two stages as a part of first order. They are

1. The engineering stage
2. The production stage

LIFE CYCLE ASPECT	ENGINEERING STAGE	PRODUCTION STAGE
Risk reduction	Schedule, technical feasibility	Cost
Products	Architecture baseline	Product release baselines
Activities	Analysis, design, planning	Implementation, testing
Assessment	Demonstration, inspection, analysis	Testing
Economics	Resolving diseconomies of scale	Exploiting economies of scale
Management	Planning	operations

The two stages is a little too course, too simplistic for more applications. Hence, the Rational Objector software engineering methodology divides one development cycle into four consecutive phases: inception phase, elaboration phase, construction phase, and transition phase. The engineering stage is decomposed into inspection and elaboration phase, where the production stage is decomposed into construction and transition phase. These four phases of the life-cycle process are loosely mapped of the conceptual framework of the spiral model and are named to depict the state of the project. In the figure, the size of the spiral corresponds to the inertia of the project with respect to the breadth and depth of the artifacts that have been developed. This inertia manifests itself in maintaining artifact consistency, regression testing , documentation, quality analyses, and configuration control. Increased inertia may have little, or at least very straightforward, affect changing any given discrete component or activity. However, the reaction time for accommodating ajar architectural changes, major requirements changes, major planning shifts, or major organizational perturbations clearly increases in subsequent phases.

ISSN 1512-1232



The Inception Phase

The inception phase establishes the business case for the system and defines the system's scope. The business case includes success criteria, risk assessment, estimate of the resources needed, and a phase plan showing dates of major milestones. At the end of the inception phase, the life cycle objectives of the project are examined to decide whether to proceed with the development.
Primary objectives:

- Establishing the project's software scope and boundary conditions, including an operational concept. Acceptance criteria an a clear understanding of what is and is not intended to be in the product
- Discriminating the critical use cases of the system and the primary scenarios of operation that will drive the major design trade-offs.
- Demonstrating at least one candidate architecture against some of the primary scenarios
- Estimating the cost and schedule for the entire project
- Estimating potential risks

Essential activities

- Formulating the scope of the project.
- Synthesizing the architecture.
- Planning and preparing a business case.

Elaboration Phase

The goals of the elaboration phase are to analyze the problem domain, establish a sound architectural foundation, develop the project plan and eliminate the highest risk elements of the project. At the end of the elaboration phase, the detailed system objectives, scope, choice of architecture, and the resolution of major risks are examined.
Primary objectives

- Base lining the architecture as rapidly as practical
- Base lining the vision
- Base lining a high-fidelity plan for the construction phase

ISSN 1512-1232

- Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time

Essential activities

- Elaborating the vision
- Elaborating the process and infrastructure.
- Elaborating the architecture and selecting components.

The Construction Phase

During the construction phase, a complete system is iteratively and incrementally developed and made ready for transition to the customer community. This includes completing the implementation and testing of the software. At the end of the construction phase, the operational decision is made.

Primary objectives

- Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework
- Achieving adequate quality as rapidly as practical
- Achieving useful versions

Essential activities

- Resource management, control, and process optimization
- Complete component development and testing against evaluation criteria
- Assessment of product releases against acceptance criteria of the vision

The Transition Phase

During the transition phase, the software is shipped to the customer. This phase typically starts with a ``beta release" of the systems. At the end of the transition phase, the life cycle objectives are reviewed and possibly another development cycle begins.

Primary objectives

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

Essential activities

- Synchronization and integration of concurrent construction increments into consistent deployment baselines.
- Deployment-specific engineering
- Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set.

4 CONCLUSION

Software Process is a compendium of information that describes how a software organization plans for, manages, builds, qualifies, and maintains software. In other words, the standard software process consists of a number of building blocks that are used in different ways to support software projects. Development process in software is unfathomable. Its limits are not difficult to be assigned nor it possible to fix boundaries. Yet, as a matter of discipline, it is necessary to tailor the process to the specific needs of one's project to one's hand. As the clock has two hands, the software development process has technical complexity and management complexity. Both these complexity must be synchronized and must act in tandem so that they tick well to show good time.

ISSN 1512-1232

As discipline is tailored and built into the process development, logic coupled with judgement is injected and the methods, techniques, culture, formality, values and organization become part of the development process.

REFERENCES

1. Pressman, Scott (2005), Software Engineering: A Practitioner's Approach (Sixth, International ed.), McGraw-Hill Education.
2. David I. Cleland, Roland Gareis (2006). Global project management handbook. McGraw-Hill Professional, 2006. ISBN 0071460454. Pp.1-4.
3. Albert Hamilton (2004). Handbook of Project Management Procedures. TTL Publishing, Ltd. ISBN 07277-3258-7.
4. Edward Kit, Software testing in the real world, Addison-Wesley publications, 2000, ed.1
5. Pankaj Jalote, An integrated approach to software engineering, Narosa publications, 1997, ed. 2
6. Shari Lawrence Peleeger, software engineering theory and practice, Pearson education, 2001, ed. 2
7. Richard Fairly, Software engineering concepts, McGraw-Hill Inc.,1985
8. Barry W. Boehm, Software Engineering Economics, Prentice-Hall Inc., 1981.
9. Diomidis Spinellis. Code Quality: The Open Source Perspective. Addison Wesley, Boston, MA, 2006.
10. Ho-Won Jung, Seung-Gweon Kim, and Chang-Sin Chung. Measuring software product quality: A survey of ISO/IEC 9126. IEEE Software, 21(5):10–13, September/October 2004.
11. Martin Stevens (2002). Project Management Pathways. Association for Project Management. APM Publishing Limited, 2002.
12. Morgen Witzel (2003). Fifty key figures in management. Routledge, 2003. ISBN 0415369770. Pp. 96-101.
13. Bjarne Kousholt (2007). Project Management –. Theory and practice.. Nyt Teknisk Forlag. ISBN 8757126038. p.59.
14. F. L. Harrison, Dennis Lock (2004). Advanced project management: a structured approach. Gower Publishing, Ltd., 2004. ISBN 0566078228. p.34.
15. Stellman, Andrew; Greene, Jennifer (2005). Applied Software Project Management. O'Reilly Media. ISBN 978-0-596-00948-9.
16. Stephen H. Kan. Metrics and Models in Software Quality Engineering. Addison-Wesley, Boston, MA, second edition, 2002.
17. Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, IEEE Computer Society Press, 2005, ISBN: 0471713457.
18. Musa, J.D, A. Iannino, and K. Okumoto, Engineering and Managing Software with Reliability Measures, McGraw-Hill, 1987
19. Dustin, Elfriede (2002). Effective software Testing. Addison Wesley. p. 3. ISBN 0-20179-429-2.

Article received: 2010-04-29