ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 4 April, 2013 Page No. 1355-1360

A Study of Fundamentals and Principles of Software Engineering Process

Niraj Nake
Department of Electronics and Telecommunication,
PRMCEAM ,Badnera-Amravati, Maharashtra, India
Email:nirajnake24@gmail.com

Abstract-Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software and the study of these approaches; that is, the application of engineering to software. A software development process is concerned primarily with the production aspect of software development as opposed to the technical aspect, such as software tools. These processes exist primarily for supporting the management of software development and are generally skewed toward addressing business concerns. Many software development processes can be run in a similar way to general project management processes. Software process is defined as a set of activities that leads to the production of a software product. Although most of the softwares are custom built, the software engineering market is being gradually shifted towards component based. There is no any ideal approach to a software process that has yet been developed. Some fundamental activities like software specification, design, validation and maintenance are common to all the process activities.

Index Terms-Software Engineering, Discipline, Strategy,Software Management, Software Development, Development Models, Software Development Life Cycle, Requirements Engineering, Quality Assurance.

I. INTRODUCTION

In today's world system developers are faced to produce complex, high quality software to support the demand for new and revised computer applications. This challenge is complicated by strict resource constraints, forcing management to deploy new technologies, methods and procedures to manage this increasingly complex environment. Often the methods, procedures and technologies are not integrated. Therefore, they achieve less than desired improvements in productivity or force management to make trade off decisions between software quality and developer efficiency. Thus the production lines have to be developed faster, too. A very important role in this development is Software Engineering because many production processes are *'computer aided* ', so software has to be designed for this production system. It seems very important to do the software engineering right and fast. So, it is necessary to understand the fundamentals and the concepts of software engineering. There are process models which provide a specific way to develop the software. We need to focus on the constraints or parameters which have a high impact on a quality of software.

II. SOFTWARE ENGINEERING

Over the years software developers have understood that software development is not merely coding. It is something which starts long before one actually starts programming and continues even after the first version of the software is delivered. Hence it consists
of number of activities in addition to programming. Software engineering as a discipline provides methods of systematically developing and maintaining that software. It can be simply stated as strategy for producing quality software or the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
Software engineering can serve the purpose of acting like an application of scientific principles to
(i) Systematic transformation of a problem into a working software solution.

(ii) The subsequent maintenance of that software after delivery until the end of its life.

So, we can define software engineering as

(i) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software i.e. the application of engineering to software.

(ii) A discipline which provides tools and techniques for developing the software in an orderly fashion.

The advantages of using software engineering for developing software are:

- Improved quality
- Improved requirement specification
- Improved cost and schedule estimates
- Better use of automated tools and techniques
- Less defects in final product
- Better maintenance of delivered software
- Well defined processes
- Improved productivity
- Improved reliability

III. FUNDAMENTALS

3.1 Software Development Life Cycle (SDLC)

The duration of time that begins with conceptualization of software being developed and ends after system is discarded after its usage, is denoted by Software Development Life Cycle (SDLC). The steps present in this cycle are shown in figure 1.

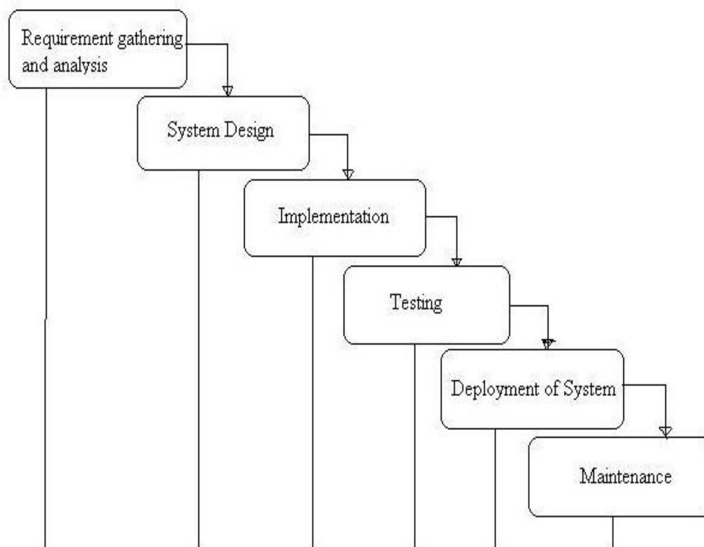

Figure 1: Steps in SDLC

3.1.1 Requirements analysis: This phase focuses on understanding the problem domain and representing the requirements in a form which are understandable by all the stakeholders of the project i.e. analyst, user, programmer, tester etc. The output of this stage is a document called Requirements Specification Document (RSD) or Software Requirements Specification (SRS). All the successive stages of software life cycle are dependent on this stage as SRS produced here is used in all the other stages of the software lifecycle.

3.1.2 System design: This phase translates the SRS into the design document which depicts the overall modular structure of the program and the interaction between these modules. This phase focuses on the high level design and low level design of the software. High level design describes the main components of software and their externals and internals. Low level design focuses on transforming the high level design into a more detailed level in terms of an algorithms used, data structures used etc.

3.1.3 Implementation: This phase transforms the low level design part of software design description into a working software product by writing the code.

3.1.4 Testing phase: This phase is responsible for testing the code written during implementation phase. This phase can be broadly divided into unit testing (tests individual modules), integration testing (tests groups of interrelated modules) and system testing (testing of system as a whole). Unit testing verifies the code against the component’s high level and low level design. It also ensures that all the statements in the code are executed at least once and branches are executed in all directions. Additionally it also checks the correctness of the logic. Integration testing tests the intermodular interfaces and ensures that the module drivers are functionally complete and are of acceptable quality. System testing validates the product and verifies that the final product is ready to be delivered to the customers. Additionally several tests like volume tests, stress tests, performance tests etc., are also done at the system testing level.

3.1.5 Deployment of System: This phase makes the system operational through installation of system and also focuses on training of user.

3.1.6 Maintenance: This phase resolves the software errors, failures etc and enhances the requirements if required and modifies the functionality to meet the customer demands. This is something which continues throughout the use of product by the customer.

3.2 Software Process Models

A software process model is a simplified description of a software process which is presented from a particular perspective. Models, by their very nature, are simplifications so a software process model is an abstraction of the actual process which is being described. Process models may include activities which are part of software process, software products and the roles of people involved in software engineering. Some

examples of the types of software process model which may be produced are:

1. A workflow model: This shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model represent human actions.

2. A dataflow or activity model: This represents the process as a set of activities each of which carries out some data transformation. It shows how the input to the process such as a specification is transformed to an output such as a design. The activities here may be at a lower-level than activities in a workflow model. They may represent transformations carried out by people or by computers.

3. A role/action model: This represents the roles of the people involved in the software process and the activities for which they are responsible.

4. Iterative Processes: This prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what they want.

5. Capability Maturity Model Integration (CMMI): This is one of the leading models and based on best practice. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. There are a number of different general models or paradigms of software development:

1. The waterfall approach: This takes the above activities and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on. After each stage is defined it is 'signed off' and development goes on to the following stage.

2. Evolutionary development: This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system which satisfies the customer's needs. The system may then be delivered. Alternatively, it may be re-implemented using a more structured approach to produce a more robust and maintainable system.

3. Formal transformation: This approach is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving'. This means that we can be sure that the developed program meets its specification.

4. System assembly from reusable components: This technique assumes that parts of the system already exist. The system development process focuses on integrating these parts rather than developing them from scratch.

3.3 Software Quality Assurance

It is defined as the Conformance to explicitly state functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. This definition serves to emphasize following important points:

- Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
- Specified standards define a set of development criteria, if the criteria are not followed, lack of quality will almost surely result.
- A set of implicit requirements often goes unmentioned. If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect

Software quality can be affected by means of the characteristics which are shown in figure 2.

- **Functionality:** It is assessed by evaluating the feature set and capabilities of the program. Functions that are delivered and security of the overall system.

- **Usability:** It is assessed by considering human factors, consistency & documentation.
- **Reliability:** It is evaluated by measuring the frequency and severity of failure, by finding accuracy of output results and by checking the ability to recover from failure and predictability of the program.

Figure 2: Characteristics of Software Quality

- **Performance:** It is measured by processing speed, response time, resource consumption, efficiency.

- **Efficiency:** It is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount.

- **Scalability:** It is the ability of the software to handle growing amount of work. For example, in web applications this is generally related to the increasing amount of web users or visitors. Alternatively this can be caused by an increasing amount of data. At some point one of the resources reaches hardware limits which are called a bottleneck. In reality, very often limit is reached because software is not using resources efficiently. Systems designed in a better way would not demand so much resource and would have an improved overall performance.

- **Extensibility:** It is the ability of a software system to allow and accept significant extension of its capabilities, without major rewriting of code or changes in its basic architecture.In software engineering, extensibility is a system design principle where the implementation takes into consideration future growth. It is a systemic measure of the ability to extend a system and the level of effort required to implement the extension. Extensions can be through the addition of new functionality or through modification of existing functionality.

- **Security:** It is a measure of the likelihood of potential security breaches due to poor coding practices and architecture. This quantifies the risk of encountering critical vulnerabilities that damage the business.
- **Maintainability:** It is the most costly phase of the software life cycle. Maintenance of the software configuration nearly always means maintenance of the procedural design representation.

3.4 Software Project Management

It is the art or mechanism of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled. We need to consider all the following management stages while managing a software project.

1. **Risk management:** It is the process of measuring or assessing risk and then developing strategies to manage the risk. In general, the strategies employed include transferring the risk to another party, avoiding the risk, reducing the negative effect of the risk, and accepting some or all of the consequences of a particular risk. Risk management in software project management begins with the business case for starting the project, which includes a cost-benefit analysis as well as a list of fallback options for project failure, called a contingency plan.

2. **Requirements management:** It is the process of identifying, eliciting, documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling the change and communicating to relevant stakeholders. Requirements management, which includes Requirements analysis, is an important part of the software engineering process; whereby business analysts or software developers identify the needs or requirements of a client; having identified these requirements they are then in a position to design a solution.

3. **Change management:** It is the process of identifying, documenting, analyzing, prioritizing and agreeing on changes to a scope of a project and then controlling changes and communicating to relevant stakeholders. Change impact analysis of a new or altered scope, which includes Requirements analysis at the change level, is an important part of the software engineering process; whereby business analysts or software developers identify the altered needs or requirements of a client; having identified these requirements they are then in a position to re-design or modify a solution. Theoretically, each change can impact the timeline and budget of a software project, and therefore by definition must include risk-benefit analysis before approval.

4. **Release management:** It is the process of identifying, documenting, prioritizing and agreeing on releases of software and then controlling the release schedule and communicating to relevant stakeholders. Most software projects have access to three software environments to which software can be released; Development, Test, and Production. In very large projects, where distributed teams need to integrate their work before releasing to users, there will often be more environments for testing, called unit testing, system testing, or integration testing, before release to User acceptance testing (UAT).

IV. PRINCIPLES

4.1 Modularity

Following the principle of modularity implies separating software into components according to functionality and responsibility. In other words, Software is divided into separately named and addressable components, sometimes called modules, which are integrated to satisfy problem requirement. Modularity is the single attribute of software that allows a program to be intellectually manageable. It leads to a “divide and conquer” strategy. It is easier to solve a complex problem. Referring figure 3, we can state that effort (cost) to develop an individual software module does decrease if total number of modules increase. However as the no. of modules grows, the effort (cost) associated with integrating the modules also grows.

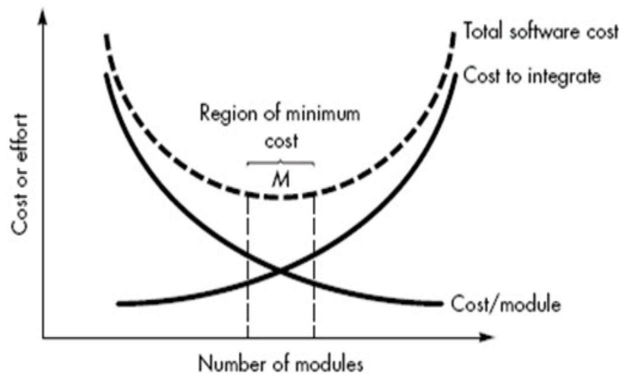

Figure 3: Effort (cost) Vs Number of modules

4.2 Abstraction

Following the principle of abstraction implies separating the behaviour of software components from their implementation. It requires learning to look at software and software components from two points of view: what it does, and how it does it. Failure to separate behaviour from implementation is a common cause of unnecessary coupling. For example, it is common in recursive algorithms to introduce extra parameters to make the recursion work. When this is done, the recursion should be called through a non-recursive shell that provides the proper initial values for the extra parameters. Otherwise, the caller must deal with a more complex behaviour that requires specifying the extra parameters. If the implementation is later converted to a non-recursive algorithm then the client code will also need to be changed.

4.3 Anticipation of Change

Computer software is an automated solution to a problem. The problem arises in some context or domain that is familiar to the users of the software. The domain defines the types of data that the users need to work with and relationships between the types of data. Software developers, on the other hand, are familiar with a technology that deals with data in an abstract way. They deal with structures and algorithms without regard for the meaning or importance of the data that is involved. A software developer can think in terms of graphs and graph algorithms without attaching concrete meaning to vertices and edges. Working out an automated solution to a problem is thus a learning experience for both software developers and their clients. Software developers are learning the domain that the clients work in. They are also learning the values of the client: what form of data presentation is most useful to the client, what kinds of data are crucial and require special protective measures. If the problem to be solved is complex then it is not reasonable to assume that the best solution will be worked out in a short period of time. The clients do, however, want a timely solution. In most cases, they are not willing to wait until the perfect solution is worked out. They want a reasonable solution soon; perfection can come later. To develop a timely solution, software developers need to know the requirements: how the software should behave. The principle of anticipation of change recognizes the complexity of the learning process for both software developers and their clients. Preliminary requirements need to be worked out early, but it should be possible to make changes in the requirements as learning progresses. Coupling is a major obstacle to change. If two components are strongly coupled then it is likely that changing one will not work without changing the other. Cohesiveness has a positive effect on ease of change. Cohesive components are easier to reuse when requirements change. If a component has several tasks rolled up into one package, it is likely that it will need to be split up when changes are made.

4.4 Generality

The principle of generality is closely related to the principle of anticipation of change. It is important in designing software that is free from unnatural restrictions and limitations. For example where the principle of generality applies, consider a customer who is converting business practices into automated software. They are often trying to satisfy general needs, but they understand and present their needs in terms of their current practices. As they become more familiar with the possibilities of automated solutions, they begin seeing what they need, rather than what they are currently doing to satisfy those needs. This distinction is similar to the distinction in the principle of abstraction, but its effects are felt earlier in the software development process.

4.5 Incremental Development

In this process, we build the software in small increments; for example, adding one use case at a time. This process simplifies verification. If we develop software by adding small increments of functionality then, for verification, we only need to deal with the added portion. If there are any errors detected then they are already partly isolated so they are much easier to correct. A carefully planned incremental development process can also ease the handling of changes in requirements. To do this, the planning must identify use cases that are most likely to be changed and put them towards the end of the development process.

4.6 Consistency

The principle of consistency is recognition of the fact that it is easy to do things in a familiar context. For example, coding style is a consistent manner of laying out code text. This serves two

purposes. First, it makes reading the code easier. Second, it allows programmers to automate part of the skills required in code entry, freeing the programmer's mind to deal with more important issues. At a higher level, consistency involves the development of idioms for dealing with common programming problems. Consistency serves two purposes in designing graphical user interfaces. First, a consistent look and feel makes it easier for users to learn to use software. Once the basic elements of dealing with an interface are learned, they do not have to be relearned for a different software application. Second, a consistent user interface promotes reuse of the interface components. Graphical user interface systems have a collection of frames, planes, and other view components that support the common look. They also have a collection of controllers for responding to user input, supporting the common feel. Often, both look and feel are combined, as in pop-up menus and buttons. These components can be used by any program.

V. CONCLUSION

In this paper, we talked about the principles and the concepts to be considered which developing a software. The obvious presence of software in every corner of the society and the ever-growing demands for high quality and secure software indicate the needs of Software Engineering as a discipline. We saw that there are many existing models for developing systems for different sizes of projects and requirements. Each model has advantages and disadvantages for the development of systems, so each model tries to eliminate the disadvantages of the previous model. Hopefully this study may stimulate the software engineering community as a whole so that we, software engineering professionals, will be able to flesh out our visions as to where we are heading from here and what we are going to do in terms of making improvement to our products in-the-small and to our civilization in-the-large.

REFERENCES

[1] Binghui Helen Wu. On Software Engineering and Software Methodologies A Software Developer's Perspective.

[2] Mary Shaw. What Makes Good Research in Software Engineering? http://www.cs.cmu.edu/~shaw/

[3] Don Gotterbarn. Reducing Software Failures: Addressing the Ethical Risks of the Software Development Lifecycle, *Australian Journal of Information Systems*

[4] Hany H Ammar, Walid Abdelmoez, and Mohamed Salah Hamdi.Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems

[5]Rainer Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering:a research survey.Journal Of Software Maintenance And Evolution: Research And Practice *J. Softw. Maint. Evol.: Res. Pract.* 2003; 15:87–109 (DOI: 10.1002/smr.270)

[6] Nishant Dubey. A Paper Presentation On Software Development Automation By Computer Aided Software Engineering (CASE). IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 1, January 2011 ISSN (Online): 1694-0814. www.IJCSI.org

[7] Nabil Mohammed Ali Munassar1 and A. Govardhan. A Comparison Between Five Models Of Software Engineering. IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010. ISSN (Online): 1694-0814 www.IJCSI.org

[8] Anthony Finkelstein , Jeff Kramer. Software Engineering: A Roadmap.

[9] Josh Dehlinger and Jeremy Dixon . Mobile Application Software Engineering: Challenges and Research Directions