EXPLORE TEN DIFFERENT TYPES OF SOFTWARE ENGINEERING PRINCIPLES

S.Thulasee Krishna[1] M.E., (Ph .D), MISTE Dr. S.SREEKANTH[2]Ph. D
B. Bhaskar[3] (M.Tech) and N.Pavan Kumar[4], M.Tech, ASST.PROF

[1 3 4]Dept of CSE, Kuppam Engineering College, Kuppam -517425, Chittoor (Dt), A.P. India
thulasi_krishna2001@yahoo.com pavan4786@gmail.com

[2]Dept Of IT, Sitams, Chittoor(Dt),A.P
pranavsree_2000@rediffmail.com

Abstract-
The aim of engineering is to repetitively produce complicated artifacts in an efficient way. Engineering is a set of disciplines seeking solutions for complicated problems and systems that could not be done by individuals. To contribute to the Software engineering discipline from an engineering perspective, through the identification of software engineering Fundamental principles and the description of operational guidelines for these engineering fundamental principles. This paper Explore on the set of fundamental principles will contribute to a better understanding and possibly from an engineering perspective.

Keywords: *Engineering, Software engineering, nature of software engineering, principles of software engineering.*

1. INTRODUCTION

Since its inception close to 40 years ago, software engineering has certainly matured considerably. While many contributors to the discipline have worked on developing methods, techniques and tools, few have worked at defining the discipline's foundations. Solidifying this foundation is now needed to enable software engineering to continue and even accelerate its maturation as a true discipline in and of itself, and, more specifically, as a legitimate engineering discipline. The search for the foundation of software engineering is needed not only by individual practitioners in the field, but by standards organizations and educators as well. Standards organizations have a mandate to develop and maintain a corpus of standards. Software engineering standards had been developed on an ad hoc basis, which sometimes led to "*standards [being]… inconsistent, overlapping, and* occasionally contradictory." In mature engineering disciplines, it is possible to audit the relationship between practice standards and the engineering principles that constrain these standards. But this relationship is less obvious in software engineering. As Moore reports, software is an intangible product which is not constrained by physical laws. Software engineering is also still in many respects an emerging discipline and some of its main concepts are not yet mature [1]. Thus, software engineering standards organizations need a better identified and recognized foundation to improve the overall quality of their standards which are increasingly being used by industry. While trying to identify what might be the foundation of the discipline, various authors over the past thirty years have looked for principles, concepts, techniques or laws underlying the field. the early '70s on the search for fundamental principles of software engineering, in terms of both methodology used.

II. ENGINEERING

"To Be or Not To Be?"

To many professionals engineering means systematic planning, teamwork, rigorous process, repeatability, efficiency. Software professionals have been arguing the term "software engineering" and its implication for three decades since Frits Bauer invented it in 1968 [1-2]. Yet, still some fundamental questions remain, such as:

a. Is software development an Engineering discipline?

b. Are software developers engineers? Or craftsmen?

There were completely different assertions and opinions on the above issues of "to be or not to be" that is still confusing the academics, practitioners, and students in software engineering and in the software industry. In investigating these fundamental problems, the authors find the myth was caused by a confusion of time in perceiving software development as, or as not, an engineering discipline. The authors' answer to the question whether software development is an engineering discipline is simply 'no.' While more precisely: it is 'not' at present and in the past, and it is going to be and should be 'yes' in the future. Currently, software development is evolving from the laboratory-oriented and all-round-programmer-based practice to an industry-oriented and process-based platform, and software developers are experiencing changes of roles from craftsmen to regulated professionals – the software engineers [2]. The practices of the former are based on personal talents, tastes and art, while those of the latter are based on disciplined processes and repeatable professional activities.

III. SOFTWARE ENGINEERING

Table 1. Analysis of Representative Definitions of Software Engineering

No.	Nature	Means	Aims	Attributes of Aims
	A method	Engineering principles	Software	- economy - reliability - efficiency
2	A science and art	Lifecycle methods: - specification - design - implementation - evolving	Programme and Document	- economy - timeliness - elegance
3	An engineering discipline	Engineering approaches: - standards - methodologies - tools - processes -quality assurance systems	Large-scale software	-productivity - quality - cost - time

Contrasting the differences between the three definitions of software engineering leads to the distinctions noted table 1. Shows how the concepts surrounding software engineering can be enhanced, particularly in terms of its means, aims and attributes; resulting in a deeper understanding of software engineering.

Some points regarding the perceived nature, means, aims and their attributes in the answers can be made. The first definition proposed software engineering as a method or approach to software development; the second definition focused on scientific methods and art for programming; and the third definition portrays software engineering as an engineering discipline for developing large-scale software in the software industry.

IV.THE NATURE OF SOFTWARE ENGINEERING

Conventional industries are producing artifacts from raw materials via engineering approaches; Software industry is producing software solutions for problems via software engineering. In 1975 Hoare identified four characteristics of software engineering – professionalism, vigilance, sound theoretical knowledge, and tools. These characteristics were quite generic for perceiving software engineering at that time. In a further development in 1996, Wasserman described eight technical characteristics of software engineering as follows:

- Abstraction
- Methods and notations
- Prototyping
- Modularity and architecture
- Lifecycle and process
- Reuse
- Metrics
- Tools and integrated environments

Wasserman's vision mainly covered the technical characteristics of software engineering. Therefore Hoare's view has been seen as a balance to show both the sights of the forest and trees in describing the young discipline of software engineering. To investigate the nature of software engineering in a systematic way, the authors find there are five categories of characteristics of generic engineering principles, which software engineering can borrow. They are the categories of engineering aim, organizational, technical, managerial, and social characteristics [3]. According to the classification, a key to software engineering is the category of engineering organization, which characterizes the following features:

- Apply systematic processes
- Support co-operative work
- Adopt division of work
- Establish standardization
- Adopt tools and machinery
- Plan actual schedule
- Optimize resources allocation
- Derive predictable outputs
- Seek controllable quality

These characteristics determine the level of maturity in engineering organization. All of them are applicable towards maturing the software engineering discipline. To further explore the nature of software engineering, the authors found it is useful to contrast the three-generation definitions of software engineering. The first definition of software engineering was provided by Bauer [2] more than three decades ago. In his paper Bauer defined software engineering as:

"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines." One of the well-accepted second generation definitions of software engineering, by McDermid [4], is as follows: "Software engineering is the science and art of specifying, designing, implementing and evolving - with economy, timeliness and elegance – programs, documentation and operating procedures whereby computers can be made useful to man." Along with the evolution of software engineering research and practices, the authors find it is useful to shift the concept of software engineering from conventional laboratory orientation to an industrial orientation. This results in a new perception on software engineering as follows: "Software engineering is a discipline that adopts engineering approaches, such as established methodologies, processes, tools, standards, organization methods, management methods, quality assurance systems, and the like, in the development of large-scale software seeking to result in high productivity, low cost, controllable quality, and measurable development schedule."

V. SOFTWARE ENGINEERING PRINCIPLES

Royce 1970

Boehm (1983) refers to Royce (1970) as initiating one of the earliest discussions on software principles. Royce (1970) presents five rules to follow in the context of large software projects in order to mitigate risks, which he named "steps". These rules are based mainly on the author's professional experience in the management of spacecraft software projects. There is no reference to an explicit methodology, or to the work of previous authors, for the identification of these steps which are formulated as rules. Royce does, however, put an emphasis on these steps as being factors critical to the success of large software projects, even though he does not explicitly use the term 'principle'.

seven basic principles are:

- Manage using a phased life-cycle plan.
- Perform continuous validation.
- Maintain disciplined product control.
- Use modern programming practices.
- Maintain clear accountability for results.
- Use better and fewer people.
- Maintain a commitment to improve the process.

Mills (1980)

In this paper "Principles of software engineering", Mills discusses the software crisis, defining software engineering as "a growing set of disciplines", and includes three (3) discipline categories: design, development and management. Mills refers to the term 'principle' only within the context of the design discipline, and specifically identifies two principles for design: structured programming and modular decomposition. Mills provides no definition of what a principle is, nor any criteria for identifying them, which leads to some ambiguity between the software principle as a rule and the nature of software engineering itself. His paper is supported by 16 references, mostly about design and programming practices. The paper is more a generic discussion of software engineering than an examination of software engineering principles.

The general principles of software engineering are:

- Part I, in which the author relates software engineering to the whole field of the system development process - system engineering hardware engineering, software engineering, and system integration. Presented briefly are overviews of the major aspects of software engineering - design, development, and management [7].

- Part II, on the software engineering program, deals with the architecture of the new discipline. Discussed is the underlying concept of the software development life cycle . Based upon this foundation are a series of formally documented practices that set forth the specifics of software design, development, and management methods, which are presented in this paper. Also presented is an educational program whereby this discipline with its principles and practices has been made teachable.
- Part Ill, on software engineering design practices, deals with activities bounded by requirements definition on one side and program implementation on the other. Three levels of design practices are defined, dealing with construction and verification of software systems, modules within systems, and individual programs. At each stage, a new level of mathematical rigor and precision for creating and evaluating software designs is introduced.
- Part IV, on software engineering development practices, discusses a methodology for translating designs into software product. The subject is treated under two main headings, cude management and integration engineering. These are rigorous methods for building the parts and integrating them into the whole software product that meets the design specifications.
- Part V, deals with the management of software engineering, which is primarily the intellectual control of the whole software engineering process. Intellectual control is brought about by a technical review strategy, a· cost Management approach, and a Project environment for effective Software development.

Lehman (1980)
Lehman points out the importance of developing a global comprehension and understanding of principles underlying the discipline, and in particular of the maintenance phase. His research goal was to discover some basic laws or fundamental truths underlying the maintenance activities. Lehman also postulates that these laws might be helpful in the development of management tools for these activities, as well as providing a foundation for improving the maintenance process. He states that we should not expect to find laws of software engineering principles which have the precision and the predictability of natural laws. By this he means that, if laws or principles of software evolution can be formulated, they will be less precise than biological laws. Lehman's work focuses on how software systems evolve in time. He analyzed data from maintenance projects over a period of seven years and observed some "regularities", or patterns, which may suggest some basic 'laws'. In fact, Lehman proposed five (5) of them, which he called 'evolution laws'. The work of Lehman is mainly based on the analysis of 'data' collected over 7 years from maintenance projects of large software systems. The suggested laws are derived from the "regularities" observed. Descriptive information about these large projects studied is not provided, nor is the nature of the data analyzed. Lehman indicates that these laws have not been validated, and that their validation may result in the modification or rejection of some of them.

Lehman was introduced five laws:

- First, we note that *Evolution* is one of Brooks' essential characteristics of software systems: the only systems that are not evolving are the dead ones. Evolution is a basic fact of software life. The fact that we have evolution on multiple levels is often overlooked in considering the fact of evolution: local versus global, component versus system, internally versus externally motivated, etc.

- Second, we note that while we have masses of data about local and component level evolution buried in our change and version management systems, we have done relatively little with that data to determine software evolution principles or theories for the underpinnings of software engineering. Principles are foundational in providing guidance in the various levels of evolution that take place – they form the bedrock of the software engineering enterprise [5].
- Third, we note that we have very little data about software systems evolution - global evolution, evolution on a large scale. For our work on FEAST [2] we have had what amounts to a wealth of data: a mere handful of systems. The utility of these few sets of data is hampered further by the fact that there are at most about 25 data points (ie, 25 systems evolution instances) for each system.
- Fourth, we note that we have in this small amount of data only a few of the important attributes needed to understand evolution deeply - namely, we have basic attributes such as system size, release dates (in some cases), etc. To substantiate laws of evolution we need more systems and instances as well as more attributes for those systems and instances.
- Fifth, we note that we need more than just (more) product data. We need process and organizational data to determine the fate of the FEAST hypothesis [3]. We need process and organizational data [4] in order to gain a deep understanding of the organizational processes and structures within which the systems are evolved. Determining the underlying correlations and causal mechanisms that show how feedback control works within these social systems to effect the evolution within the software systems requires broader data that we currently have or currently are considering.

Boehm (1983)
Boehm is the first author to have referred, in an explicit manner, to the search for ‘basic principles’ of software engineering. He analyzed historical data from multiple projects of the TRW Defense System Group to extract such basic principles for the success of a software project. On the basis of his analysis, Boehm identifies seven (7) ‘basic independent principles’ of software engineering. He is also the first to have defined two (2) criteria for identifying principles. First, the principles should be independent of each other, that is, the use of two principles cannot generate a third one. Second, the entire space should be “represent able” by combinations of the basic principles. Boehm states that, while these principles do not answer all the questions, they do provide a base from which to work. His analysis is supported by 49 references, making his work the most completely documented on the subject up to that time. Each of the principles is further described in Boehm's text to give the reader some background and a better understanding of their meaning. Although he does not formally define the term ‘principle’, the principles are formulated as rules to follow. Even though Boehm reports that he studied over 30,000,000 person-hours of software development to generalize his set of seven principles, his research methodology is not documented [6].

Boehm introduced seven principles of software engineering:

- Principle 1. Separation of Concerns.
- Principle 2. Modularity.
- Principle 3. Incrementality.
- Principle 4. Abstraction.
- Principle 5. Generality.

- Principle 6. Anticipation of Change.
- Principle 7. Rigor and Formality**.**

Davis (1995)

In 1995, Davis published a guidebook on software development principles. His book identifies 201 'principles' as a guide for engineers, managers, students and researchers in software engineering. Davis points out in particular that, while there are many books and papers on methodologies, techniques and tools for developing software, there is a scarcity of resources on software engineering principles. Davis is the first to propose a definition of the term 'principle' as *"a basic truth, rule or assumption about software engineering that holds regardless of the technique, tools, or language selected."* He also states that, if software engineering is truly an engineering discipline, then its definition should be:

"The intelligent application of proven principles, techniques, languages and tools to the cost-effective creation and maintenance of software that satisfies user's needs." He refers to the following previous authors on this topic: Royce, Lehman and Boehm. He also mentions that software engineering cannot be based on natural laws, as is the case for the classical Engineering disciplines. Davis is therefore of the opinion that software engineering must evolve its own principles based mainly on the observation of projects. Later discussed a search for a stable base on which to establish the foundation of software engineering, Davis introduces the concept that a 'principle' evolves over time, and that some new principles will be added while others will be removed. In his view, the list of principles will follow the evolution and the transformation of the software engineering discipline. He classifies his 201 principles into eight categories corresponding to software development phases, and then subsequently chooses 15 as being the most important principles of software engineering. For each of his principles, Davis provides a reference to an article written by practitioners or researchers. His book contains 124 references supporting his set of principles. Moreover, each principle is commented on through its relationship to other proposed principles. Davis did not analyze these principles to generalize them into a smaller set, as was done by Boehm and later by Bourque *et al.* Moreover, the principles might not all be independent, as were those in Boehm's set, and also some principles may be contradictory. He states: "I make no claim that these 201 principles are mutually exclusive… a combination of some of these principles may imply another". Davis is the first to assert that principles are not as stable as inferred by other authors. Moreover, he does not describe any criterion for the identification of his set of principles, nor does he describe how he chose these principles from among other candidates. Each principle is formulated as a rule, as is the case with Boehm [8].

Davis introduced fifteen software engineering principles are:

- make quality number
- High quality software is possible
- Give product to customer early
- determine the problem before Writing the requirement
- evaluate design alternatives
- use an appropriate process model
- use different languages for different phase
- minimize intellectual distance
- put technique before tools
- get it right before you make it faster

- inspect code
- good management is more important than good technology
- people are the key to success
- follow with care
- take responsibility

Wiegers (1996)

Wiegers' goal is to identify what is required to develop a software engineering culture in an organization in order to increase the quality and the efficiency of the software engineering process and the resulting software products. To reach this goal, Wiegers states that organizations need to establish or modify the organizational software culture, including a set of values, objectives and principles guiding individuals, activities, priorities and decisions in organizations. He identifies 14 software engineering principles which have an influence on the software engineering culture of an organization. These principles correspond to the basis of the cultural changes that had been experienced at Kodak by the author. Each of Wiegers' principles is formulated as a rule to follow, as was the case with Boehm and Davis. Wiegers does not define the term 'principle', nor the criteria used to identify one, but states that principles help select development practices which improve software development processes and products, and must begin by establishing or modifying a software engineering culture in the organization.
Some principles of software engineering are:

- A principle is a proposal formulated in a prescriptive way;
- A principle should not be directly
- Associated with, or arise from, a Technology, a method, or a tech- Unique, or itself be an activity of Software engineering;
- The principle should not dictate a compromise (or a proportioning) between two actions or concepts;
- A principle of software engineering should include concepts connected to the engineering discipline; It must be possible to test the formulation of a principle in practice, or to check its consequences [9].

Wasserman (1996)

Wasserman observes that, although there are rapid changes in software development technologies, some fundamental ideas or concepts seem to remain stable, thereby providing a viable foundation for the software engineering discipline. He states that these fundamental concepts have close relationships; thus, they are not independent, as were those proposed by Boehm. Wasserman also states that these concepts form the basis of the discipline's best practices. He also notes that there are some risks to ignoring these fundamentals concepts, including:

· Development errors;
· High maintenance costs;
· Failure to build software that meets
Customer's requirements.

Wasserman does not define precisely what is meant by a 'fundamental concept', nor by the criteria and methodology for choosing them. He also uses a variety of terms such as concept, technique, notion, tool and method interchangeably, and his fundamental concepts are not formulated as rules.
Wasserman introduced some principles:

- According to Wasserman, would SE principles practiced in the USA be the same as those practiced in Japan? The most important aspect of your answer is the why or why not?
- Define Wassermann's notion of *Modularity and Architecture* and discuss why its important (mention design patterns and how that relates to standardization trends).
- What are the five partitioning approaches Wasserman mentions in his section on *Modularity and Architecture*? Do you agree or disagree with his stance on these? Why?
- What are the roles of Reuse and Metrics in the *Life cycle* and *Process* according to Wasserman? Name and define at least two product metrics and two process metrics [10].
- What are the five main issues surrounding Software Engineering Environments? Choose the most fundamental issue and discuss why you think its important (i.e., more important than the other four)?

Maibaum (2000)

Maibaum (2000) is of the opinion that, if software engineering is truly an engineering discipline, then it must have mathematical foundations, as is the case with other mature engineering disciplines. He states that software is based on radical design, and that software development is still largely a custom-made product. Maibaum bases this conclusion on Vincenti's work, and is of the opinion that software should follow the normal design process by using more ready-to-use components. Hence, he Presents modularization as the only "method" for dealing with the ever-growing software complexity, recommending that more studies be done on developing "behavioral principles" for concurrent processing. Also, he states that "*software engineering is not that good on adopting engineering principles of measurement in data gathering.*" Maibum neither defines the foundations of the software engineering discipline nor its mathematical foundations, but rather postulates a list of theoretical and methodological issues (inspired by Vincenti's categories of engineering knowledge) as candidate contributions to better software engineering.

Meyer (2001)

Meyer is of the opinion that, even though no definition of software engineering has received general consensus, educational institutions have the responsibility for training future software professionals to develop software products which will satisfy the customer. From this perspective, Meyer states that the first (of five) components of software engineering curricula are 'principles'. He defines a principle as *"a long* lasting concept that underlies *the whole* discipline.*"* The terms 'principle' and 'concept' seem to be synonymous for Meyer. He also mentions that the principles have not really changed since the emergence of the discipline, in contrast to Davis' opinion on the evolution of software engineering principles. Meyer points out that these principles are not based on techniques, but are more a mode of thinking, a kind of intellectual framework. Meyer states that these principles are an important part of the knowledge that educators must convey to their students.

Meyer introduced some principles are:

- Abstraction: separate essential from the auxiliary [11].
- Distinction between specification and implementation: confusing in software.
- Recursion: apply definition to some of its parts: classes, grammars, functions, etc.
- Information hiding: what you export and what you hide. Reuse: when to rely on someone else's job.
- Battling complexity: recognize simplicity in an apparent mess.
- Scaling up: which techniques will scale up?

- Designing for change: change process can be painful, especially for large systems.
- Classification: class hierarchies.
- Typing: study of type systems for safe construction of software.
- Contracts: pre and post conditions and invariants.
- Exception handling.
- Errors and debugging.

GUAY (2004)
"Fundamental principles of software engineering" are defined in (Guay 2004) as "knowledge or elementary rules that constitute the foundation and the essence of software engineering." The following criteria were developed in the course of this research for the recognition of fundamental principles of software engineering [12].
Guay proposed some principles are:

- Fundamental principles are less specific than methods and techniques, i.e. specific methods and techniques may be selected, within a particular technological context, to accomplish the intent of fundamental principles;
- Fundamental principles are more enduring than methods and techniques, i.e. fundamental principles should be phrased in a way that will stand the test of time, rather than in the context of current technology;
- Fundamental principles are typically discovered or abstracted from practice and should have some correspondence with best practices; · Software engineering fundamental principles should not contradict more general fundamental principles;
- A fundamental principle should not conceal a trade-off. By that we mean that a fundamental principle should not attempt to priorities or select from among various qualities of a solution; the engineering process should do that. Fundamental principles should identify or explain the importance of the various qualities among which the engineering process will make trades;
- but, there may be trade-offs in the application of fundamental principles;
- A fundamental principle should be precise enough to be capable of support or contradiction;
- A fundamental principle should relate to one or more underlying concepts.

VI. CONCLUSION

This paper has reported on a series of efforts undertaken to try to identify a set of fundamental principles of software engineering. But so many authors introduced software engineering principles in different ways. As software development professionals, you need knowledge of specific technologies to do your job. If you need knowledge of software engineering principles to do your job well. A continuing pursuit of such knowledge is one mark of a true professional. This paper explores different principles; this is useful to software developers.

VII. REFERENCES

[1]. Abran, A.; Moore, J.W.; Bourque, P.; Dupuis, D. (2002). *Software* Engineering Body of Knowledge. IEEE..

[2]. Bauer, F. L. (1976), Software Engineering, in Ralston, A. and Meek, C. L. (eds.), Encyclopedia of Computer Science, Petrocelli/Charter.

[3]. Wang, Y. and Graham, K. (2000), Software Engineering Processes: Principles and Applications, CRC Press, USA, ISBN: 0-8493-2366-5, pp. 1-746.

[4]. McDermid, J. A., ed. (1991), Software Engineer's ReferenceBook, Butterworth-Heinemann Ltd., Oxford.

[5].Lehman, M. (1980). On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. *Journal of Systems and Software*, July, vol.1, no. 3, 213-221.

[6].Pankaj kamthan. software engineering principles, journals ,pp.03 . *[7].H.D.Mills,(1980). The management of software engineering Part 1: Principles of software engineering.IBM journals, vol.19,no.3-4.*

[8]. Davis, A.M. (1994). Fifteen Principles of Software Engineering. *IEEE Software*, November, 94-101. [9].Wiegers, K.E. (1996). *Creating a Software Engineering Culture*. New-York : Dorset House Publishing.

[10].Wasserman, A.I. (1996). Toward a Discipline of Software Engineering. IEEE Software, November, 23-31.

[11].Meyer, B. (2001). Software Engineering in the Academy. *IEEE Computer*, May, 28-36.

AUTHORS:

S.THULASI KRISHNA received the B.Tech. Degree in Computer Science and Engineering from Jawaharlal Nehru University ,Hyderabad, India in 2005, M.E from Sathyabama University Chennai ,and Ph.D pursing from Rayalaseema University ,Kurnool. He joined as Asst.professor in VIST Engineering College in august 2005, Hyderabad. He was worked as Asso.Professor in Vidyanikethan Engineering College Tirupathi. And currently working as Associate Professor in Kuppam Engineering College. He is member of International Association of Engineering.

Dr. S. SREEKANTH has obtained Ph.D. Degree from S.V.University, Tirupathi. He is working as a Professor & Director in MCA Department, SITAMS, Chittoor, Andhra Pradesh, with the experience of 16 years. He has published 13 research papers both in international and national journals of computer science.

[illegible]

B.Bhaskar received the B.Tech. Degree in Information Technology from Jawaharlal Nehru University, Anantapur in 2010. Pursing M.Tech in Computer Science Engineering from Jawaharlal Nehru University, Anantapur.

N.PAVAN KUMAR received the B.Tech. Degree in Computer Science and Engineering from Jawaharlal Nehru University, Hyderabad in 2007, M.Tech from Jawaharlal Nehru University, Anantapur in 2011 and currently working as Assistant Professor in Kuppam Engineering College. He is member of IAENGG, MIACSIT.