The Search for Software Engineering Principles: An Overview of Results

Alain Abran [1], Normand Séguin, Pierre Bourque, Robert Dupuis

Abstract. A list of fundamental principles of software engineering is viewed as needed to solidify the foundations of the field, thereby enabling and hastening the maturation of the discipline. Most of the authors who have investigated software engineering principles note that the discipline has a foundation. Some talk about principles, others about concepts, laws or notions, while they all agree with the view that a stable basis for the discipline is formed by their own individual set of these. However, this paper illustrates that there is a clear lack of consensus about which of the proposed principles are indeed fundamental. Furthermore, authors do not share the same definition of software engineering, nor do they share the same definition of the term 'principle'. In summary, over 250 statements on what is meant by a principle are inventoried, and most are based only on the author's opinion or point of view. Therefore, significant effort is still required to pursue this research topic relating to the foundation of software engineering. In particular, more work is required to design an appropriate research methodology, including precise definitions of the terms being used.

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. Moore (1998) also points out that, until fairly recently, 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. 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. This paper presents a survey and an analysis of the work carried out beginning in the early '70s on the search for fundamental principles of software engineering, in terms of both methodology used and the status of results to date. Section 2 presents the set of individual views documented in the literature, from the early 1970s to the late 1990s. Section 3 presents more recent collective research work conducted in the 1990s and the early 2000s. A summary of key insights is presented in section 4, and a discussion and suggestions for further work in section 5.

Individual views

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".

[1] Alain Abran and Pierre Bourque, Department of Software Engineering, École de Technologie Supérieure, 1100, rue Notre-Dame Ouest, Montréal , Canada H3C 1K3, aabran@ele.etsmtl.ca, pbourque@ele.etsmtl.ca
Normand Séguin and Robert Dupuis, Department of Computing, Université du Québec à Montréal, CP 8888 succ. CV, Montréal, Canada, H3C 8C8, seguin.normand@uqam.ca, dupuis.robert@uqam.ca

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, nor 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'.

Mills (1980)

In his 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.

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.

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 "representable" 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.

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. Although other authors like Ghezzi *et al.* (2003) 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"[p.xi].*

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.

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.

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.

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 is '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.

Although Meyer does define the word 'principle', there is still some ambiguity in this definition because of the use of the term 'concept', which does not have exactly the same meaning. Consequently, this proposed list of principles contains some statements about principles which are rules to follow, while others refer to concepts only and some that look like expected (or desired) characteristics of a programming language. How this set of principles was identified is not documented, nor are the criteria that were used to do so.

Collaborative research work

Booch and Bryan (1994)

Booch and Bryan (1994) propose a definition of software engineering that includes the term 'principle': *"Software engineering is the application of sound engineering principles to the development of systems that are modifiable, efficient, reliable, and understandable."*

To reach these quality goals, Booch and Bryan place the principles at the base of methods and methodologies. However, they indicate that principles by themselves are not sufficient to ensure that the expected quality goals of given software are achieved. Methods are defined as a disciplined process for producing software artifacts, and methodologies as a collection of methods to develop software.

Booch and Bryan propose seven (7) 'basic principles' of software engineering relating to the four quality goals specified in their definition of software engineering. They also emphasize that humans are limited in the number of concepts they can handle at a time. Therefore, the first four principles deal with software complexity. They also state that although these first four principles are important, they are not sufficient to guarantee that the software produced is functional and correct. Therefore, three additional principles are proposed.

Although Booch and Bryan do not explain explicitly how they chose their principles, they do describe the relationship of these principles to the software quality goals: these quality goals could then be thought of as the criteria used to select the principles. Finally, Booch and Bryan are very focused on programming language issues, with the ADA language supporting all their principles. This might also be considered as a criterion for selecting principles.

Ghezzi *et al.* (1991;2003)

Ghezzi *et al.* published a software engineering textbook on software 'principles'. In their book, they explicitly relate software engineering practices to the corresponding principles that they have identified. The first chapter explains how the principles underlie the discipline, and the following chapters illustrate how these principles can be applied within each phase of the software life cycle, independently of the life cycle model chosen or the technologies used.

Ghezzi *et al.* adopt the IEEE Std 610.12-1990 definition of software engineering. They postulate that principles are the foundation of software engineering: "…principles that we believe are essential to the multi-person construction of multi-version software."

Moreover, they assert that principles are more important than methodologies or tools, and that such principles provide the engineer with the knowledge required to evaluate which methodologies to choose in the particular context of a given project. They also assert that "[as] methods and techniques will evolve… principles, on the other hand, will remain more stable; they constitute the foundation upon which all the rest may be built."

Ghezzi *et al.* define the term 'principles' as "…general and abstract statements describing desirable properties of software processes and products." It is important to note that this definition makes a distinction between the process and the product. Furthermore, they indicate that principles alone are not sufficient to develop good software: the engineer also needs methods and techniques for applying principles.

The principles at the heart of the model are considered to be more stable than the tools used to develop software at the outer edge of the circle. Ghezzi *et al.* chose these principles according to two quality goals: reliability and evolvability. In their book, they state, as do other authors, that principles provide a more stable and more durable basis for the discipline than methods and techniques which are closely related to tools: "*…they [principles] constitute the foundation upon which all the rest may be built.*" The term 'rest' refers to the methods and techniques and the methodologies and tools that are required to develop and maintain software.

Twelve years later, in their second edition (2003), Ghezzi *et al.* continue to stress the importance of principles, though they have considerably updated the content of the chapters dealing with tools and techniques. They make the following statement in the 2003 preface: "*We are pleased to find that the premise of the book, the durability and importance of principles, has been borne out through the passage of time… principles of software engineering have remained the same.*" *(check quote)*

Ghezzi *et al.* highlight that the proposed principles are not specific to software engineering: "…these principles are, first of all, engineering principles." Also, Ghezzi *et al.* suggest that there might be a hierarchy of principles. They point out that the principle "anticipation of change" might be the one that distinguishes the software engineering discipline from other engineering disciplines.

The Ghezzi *et al.* principles are not formulated as rules to follow, nor do the definitions given include the term 'rule'. Moreover, Ghezzi *et al.* do not define what they mean by the term 'fundamental'.

Bushman *et al.* (1996)

In their book on software patterns, Bushman *et al.* point out that software construction is based upon "fundamental principles". They used the terms 'principle' and 'enabling technique' interchangeably, arguing that the techniques were developed to implement the principles, and that it is difficult to differentiate between those principles and the techniques that embed them.

Bushman *et al.* assert that these principles or enabling techniques are independent of methodologies and technologies. Also, they point out that most of the principles are not new, but have been known since the 1970s, notably in publications on structured programming.

The authors present eleven (11) principles as being the most important ones for software architecture. They state that the principles are not complementary and some even may be contradictory, while others are very closely related to each other, such as the principles of abstraction and encapsulation.

The term 'principle' is not defined explicitly, nor are the criteria for identifying one. Principles are not formulated as rules to follow like those presented in Boehm and Davis, and the scope of the candidate principles is restricted to software architecture. It is also claimed that these principles are "widely accepted".

Dupuis *et al.* (1997), Jabir (1998), Bourque *et al.* (2002)

In 1996, a collaborative study was initiated to identify fundamental principles of software engineering. In contrast to previous published work on software engineering principles, the research methodology of this project is documented. It includes, for instance, the adoption of a widely known definition of software engineering from IEEE standard 610.12-1990:

"(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software.
(2) The study of approaches as in (1)"

In Jabir (1998), the authors set forth four characteristics of software engineering principles:

- Any particular statement of a fundamental principles is imperfect;

- At any time, some fundamentals principles are tacit;
- Software engineering principles are derived from inheritance from other disciplines;
- Software engineering may have unique principles.

Also identified are seven criteria for accepting a candidate as a principle:

1. Fundamental principles are less specific than methodologies and techniques;
2. Fundamental principles are more enduring than methodologies and techniques;
3. Fundamental principles are typically discovered or abstracted from practice and should have some correspondence with best practices;
4. Software engineering fundamental principles should not contradict more general fundamental principles, but there may be tradeoffs in the application of principles;
5. A fundamental principle should not conceal a tradeoff;
6. A fundamental principle should be precise enough to be capable of support or contradiction;
7. A fundamental principle should relate to one or more underlying concepts.

In addition, a model of the role and the relationship between principles, standards and practices is proposed – see Figure 1. This model is adapted from the one presented in Moore (1998).

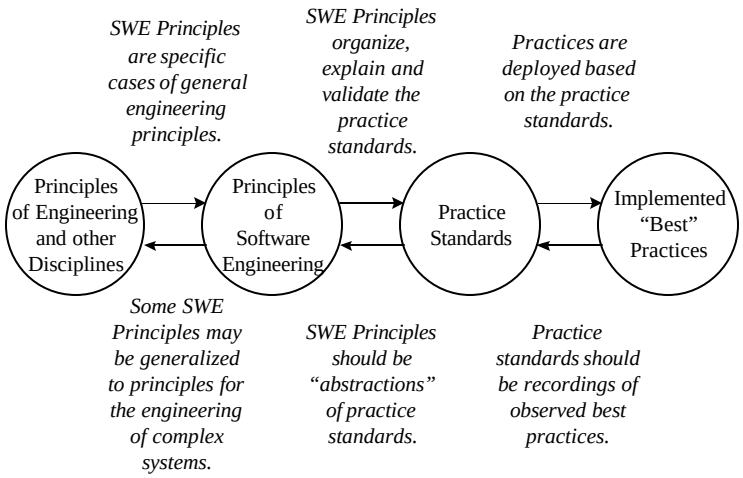

Fig. 1. Relationship between principles, standards and practices (Bourque *et al.* 2002)

A definition of the term 'principle' is proposed: "*In contrast [to concepts], fundamental principles are to be regarded as engineering statements which prescribe constraints on solutions to problems, or constraints on the process of developing solutions.*"

Although this definition does not explicitly include the term 'rule', the terms 'constraint' and 'prescribe' are strongly related to the concept of a rule. While this definition explicitly includes the software engineering process, it also implicitly refers to the software product in referencing 'solutions' to problems or constraints. Thus, this definition refers to two key concepts of software engineering: process and product. In the list of proposed candidate principles, no distinction is made between process-oriented principles and product-oriented principles. Moreover what is meant by fundamental principles, it is not explained: for instance, are there some principles that are more fundamental than others?

The research methodology included two IEEE-sponsored workshops and two Delphi studies (of 2 or 3 rounds), followed by a Web-based survey among members of the Technical Council on Software Engineering of the IEEE Computer Society. During these stages, more than 560 people were involved in one or more of the research steps. This study is the most comprehensive empirical study to date to have investigated software engineering principles. As a result of this research (methodology?), a list of 15 candidate fundamental principles is proposed.

In contrast to other studies where lists of principles have been proposed based on the opinion of one person, usually the author, this list of 15 candidate fundamental principles was identified based upon expert opinions of the participants in the two Delphi studies. The degree of consensus for each candidate fundamental principle is also documented. Furthermore, this group considers their work to be an exploratory step in the search for fundamental principles, and formally recognizes the inherent limitations of the methodology by referring to their findings as a list of "candidate" principles.

Insights gained

An overview of the key characteristics of the reviewed literature is presented in Table 1: for each reference, the terminology used is presented, whether or not a definition of the term used for the principle is indicated, and the criteria for recognizing the principles are provided. Table 1 also indicates the number of proposed principles, the statement style, and whether or not the proposed list is supported by opinions, observation, historical data or group opinions.

Table 1. Key characteristics of publications about software engineering principles (in alphabetical order)

Reference	Terms[2]	Definition	Criteria	Number	Statement Style	Source
Boehm (1983)	Principle	None	Yes (2)	7	Rules	Historical data analysis
Booch & Bryan (1994)	Principle	None	No	7	Concept	Literature
Bourque al. (2002)	Principle	Yes	Yes (8)	15	Rules	Expert opinions
Buschman et al.*et al.* (1996)	Principle/ Technique	None	No	10	Concept	Literature
Davis (1995)	Principle	Yes	No	201	Rules	Literature
Ghezzi et al.*et al.* (2003)	Principle	Yes	No	7	Mix	Literature, opinion
Lehman (1980)	Laws	None	No	5	Concept	Observation, analysis
Maibaum (2000)	Principle	None	No	3	Concept	Opinion
Meyer (2001)	Principle	Yes	No	13	Mix	Opinion
Mills (1980)	Principle	None	No	4	Concept	Opinion
Royce (1970)	Steps	None	No	5	Rules	Opinion
Wasserman (1996)	Concept	None	No	8	Concept	Opinion, literature
Wiegers (1996)	Principle	None	No	14	Rules	Observation, opinion

Term and definition

The term 'principle' is referred to in 10 of the 13 references included in this literature review, though it is interpreted variously. Buschman (1996) uses the terms 'principle' and 'enabling techniques' as synonyms, while Lehman (1980) uses the term 'laws' by analogy with laws that underlie classical engineering disciplines. Wasserman (1996) uses the terms 'concepts' and 'fundamental ideas' that underlie the discipline. Although several authors used the term 'principle', only four references provide a definition of the term 'principle', again with some differences. This lack of an explicit definition of the term 'principle' (9 of 13) has led to some ambiguity in the use of this term: while the terms 'principle', 'concept', 'technique' and 'notion' have often been used interchangeably, they are obviously not equivalent.

Moreover, the formulation of the principles varies from rules to follow to concept statements such as "abstraction". This lack of explicit definitions might be what led to this wide range of formulations.

In addition, Bourque *et al.* (2002) and Ghezzi *et al.* (2003) use the qualifier "fundamental" to describe principles, while neither define what they mean by the term.

[22] Term : Which term is used in the reference for naming the principle?
Definition : Does the reference provide a definition for the principle or the equivalent?
Criteria : Does the reference provide criteria for identifying principles?
Number : Number of principles identified
Statement Style : Formulation of the principle statements
Source : Where does the principle come from?

Criteria

Only 2 of the 13 references explicitly identify criteria for selecting and formulating software engineering principles: Boehm (1983) identifies two criteria (principles should be independent and should cover the whole discipline), while Bourque *et al.* (2002) define eight different criteria. Only Boehm (1983) emphasizes the independence concept among principles, while others said (explicitly or implicitly) that proposed principles are not independent, and some might even be contradictory.

Both Boehm (1983) and Bourque *et al.* (2002) produce a limited list of principles, through a generalization step in their methodology. Boehm (1983) made a generalization effort in order to achieve his short list of independent principles. Bourque *et al.* (2002) use a research methodology with two iterations of Delphi studies and review steps through two workshops to reduce a proposed initial list of 65 proposals to a shorter list of 15 candidate principles.

Research Methodology

An overview of the research methodology followed to identify the list of principles for each reference is presented in Table 2, with an indication of publication type (paper or book), the basis for discussion (theoretical or empirical), the type of research methodology, if any, the number of supporting references, and the scope of the proposed references (e.g. architecture, maintenance, curriculum, entire life cycle, etc.)

Many of the references proposing a list of software engineering principles are not supported by a research methodology: only 7 of the 13 references refer to a methodology supporting the identification of the proposed principles. However, 5 of these 7 references refer to an implicit methodology which is neither formally presented nor discussed by the authors. For example, Boehm (1983) notes that over 30,000,000 person-hours of effort in software projects were analyzed, but no supporting evidence is provided for independent review and analysis. Only Bourque *et al.* (2002) refer explicitly to a research methodology about how the authors conducted their exploratory study, including the relevant definitions, criteria and documented research methodology steps, as well as all intermediate research deliverables.

Table 2. Classification of references (in alphabetic order)

Reference	Publication Type	Discussion	Research Methodology	Supporting Number of References	Scope
Boehm (1983)	Paper	Empirical	Implicit	49	Life cycle
Booch & Bryan (1994)	Book	Theoretical	Implicit	12	Construction
Bourque al. (2002)	Paper	Empirical	Explicit	11	Life cycle
Buschman et al. (1996)	Book	Theoretical	-	10	Architecture
Davis (1995)	Book	Theoretical	Implicit/analytic	124	Life cycle
Ghezzi al. (2003)	Book	Theoretical	Implicit	24	Life cycle
Lehman (1980)	Paper	Empirical	Implicit/ observation	13	Maintenance
Maibaum (2000)	Paper	Theoretical	-	11	General
Meyer (2001)	Paper	Theoretical	-	10	Curriculum
Mills (1980)	Paper	Theoretical	-	16	General
Royce (1970)	Paper	Theoretical	-	0	Life cycle
Wasserman (1996)	Paper	Theoretical	-	19	General
Wiegers (1996)	Book	Theoretical	Experimentation		Software Engineering Culture

Statement style

The references use different styles for formulating principles: 5 of the 13 formulate them as rules to follow, 6 of the 13 formulate them as concept statements, while the other 2 use a mix of both. These differences may come from the lack of a definition of the term 'principle' in the references. Within the references that formulate principles as concepts, none of them defines either the term 'principle' or the term 'concept'.

Source

Over 50% of the publications to date on software engineering principles are based solely on the author's opinion; only 4 of the 13 references refer to some empirical data. In general, the sources of principles are opinions (62%), literature references (23%) and analysis of data about real projects, but with little documented evidence (15%).

Scope

The reviewed references do not all have the same scope. As indicated in Table 2, some proposed sets of principles span the entire life cycle, while others are focused on a single phase, such as architecture (Buschman, 1996), software construction (Booch, 1994) or maintenance (Lehman, 1980). Davis (1995) classifies the 201 proposed principles in eight categories. These categories cover all life cycle phases. Ghezzi et al. (2003) asserts that principles are to be independent of the selected life cycle model.

Discussion, Limitations and Future Work

This literature review on the subject of software engineering principles enables a better understanding of the relevance, the strengths and the weaknesses of the reviewed references to date within this research theme.

The literature review shows that:

- Relatively few authors have investigated the issue of software engineering principles over the past thirty-five years, compared with the very large number of publications on methods, techniques and tools that have been produced. This is especially true for references covering the entire life cycle;
- There is a consensus in the references that principles should form the basis of the discipline and that the definition of a stable core which is independent of methods, techniques and tools is desirable;
- Only 4 references explicitly define what they mean by 'principle';
- Only 2 references define criteria for recognizing a software engineering principle;
- For nearly all references, the source of the proposed software engineering principle is the personal opinion of the author, and very few references support their proposals with data from past projects;
- The only publication that has an elaborate multi-phase and documented research methodology to identify and to build consensus on the set of proposed principles is Bourque *et al.* (2002).

However, the fact remains that the proposed set of fundamental principles in Bourque *et al.* (2002) was developed based on domain experts' opinions. While developing and documenting the level of consensus on the process output, this type of research has inherent methodological limitations, which must be addressed in the future. We see further work as a two-step process.

First, additional work needs to be done on the definition of fundamental principle of software engineering, and of the criteria used to identify them. The criteria used so far, described in Bourque *et al*. (2002) need to be refined and linked to other fields. We will further investigate the link between software engineering and other types of engineering, and between engineering and science in order to better characterize the field. Work has begun on this aspect by examining how other fields of science define principles and criteria. We have adopted as an intermediate conclusion that principles link concepts and are oriented towards action. Further investigation will confirm or not the value and usability of this orientation for criteria. These criteria will then be used to screen the list of candidate principles described and in other publications.

In the second step, the resulting list of principles will undergo further investigation. Techniques other than opinion surveys should be used to validate the list. We propose to conduct empirical designs for corroborating these principles both with current theories proposed in the field of software engineering and with observation of their implementation in currently recommended best practices. In particular, the set of proposed principles must be investigated to see whether or not they provide useful and substantial contributions to the successful solution of real problems of significant size and scope.

The interaction between the proposed fundamental principles and the more generic principles of engineering is also an issue that must be investigated. However, fundamental principles are often tacit in the more mature engineering disciplines. A structured comparison of the work by Vincenti (1990) with the list of proposed principles for software engineering is therefore seen as an interesting avenue for addressing this issue. Vincenti (1990) proposes a taxonomy of engineering knowledge based on the historical analysis of five case studies in aeronautical engineering covering a roughly fifty-year period.

Through a judicious combination of these proposed next steps, it is hoped that better guidance will be available on how to interpret the candidate principles, that the potential flaws of the opinion-based studies will be pinpointed and that the set of candidate principles in Bourque *et al.* (2002) can in due course be judged on the basis of usability, relevance, significance and usefulness.

References

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

Boehm, B.W. (1983). Seven Basic Principles of Software Engineering. *The Journal of Systems and Software*, vol.3, no. 1, May, 366-371.

Booch, G.; Bryan, D. (1994). *Software Engineering with Ada*. (3rd ed.). California: Benjamin/Cummings Publishing.

Bourque, P.; Dupuis, R.; Abran, A.; Moore, J.W.; Tripp, L.L.; Wolff, S. (2002). Fundamental Principles of Software Engineering - A Journey. *Journal of Systems and Software*, 2002.

Bushcmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. (1996). *Pattern-Oriented Software Architecture*. England: Wiley.

Davis, A.M. (1994). Fifteen Principles of Software Engineering. *IEEE Software*, November, 94-101.

Davis, A.M. (1995). *201 Principles of Software Development*. New-York: McGraw-Hill.

Dupuis, R.; Bourque, P.; Abran, A.; Moore, J.W. (1997). Principes fondamentaux du génie logiciel: Une étude Delphi in Le génie logiciel et ses applications. *Dixième(s ?) journées internationales (GL97)*, Paris, 3-5, décembre.

Ghezzi, C.; Jazayeri, M.; Mandrioli, D. (2003). *Fundamentals of Software Engineering*. (2th ed.). New-Jersey: Prentice Hall.

Gilb, T. (1988). Principles of Software Engineering Management. Addison-Wesley.

Hoffman, D.M.; Weiss, D.M. (2001). *Software Fundamentals – Collected Papers by David L. Parnas*. Addison-Wesley

IEEE (1990), "Standard Glossary of Software Engineering Terminology," 610.12.

Jabir; Moore, J.W.; Abran, A.; Bourque, P.; Business Planning Group?; Dupuis, R.; Hybertson, D.; Jacquet, J.-P.; Köller, A.; Lowry, E.; Tripp, L.L. (1998), A Search for Fundamental Principles of Software Engineering – Report of a Workshop conducted at the Forum on Software Engineering Standards Issues, Montréal, Quebec, Canada, 21-25 October 1996. *Computer Standards and Interfaces*, vol. 19, no. 2, March, pp. 155-160.

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.

Maibaum, T.S.E. (2000). Mathematical foundation of software engineering. *Proceedings of the Conference on the Future of Software Engineering*. ACM, 161-172.

McConnell, S. (1999). Software Engineering Principles. *IEEE Software*, March/April, vol. 16, no. 2, 6-8.

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

Mills, H.D. (1980). The management of software engineering: Part I: Principles of software engineering. *IBM Systems Journal*. Vol. 19, no. 4, 414-420.

Moore, J.W. (1998). *Software Engineering Standards – A User's Road Map*. IEEE Computer Society.

Royce, W. (1970). Managing the development of Large Software Systems. *Reprinted in 9th International Conference on Software Engineering, IEEE Computer Society Press*, 328-338.

Vincenti, W.G. (1990), *What Engineers Know and How They Know It - Analytical Studies from Aeronautical History*, Johns Hopkins, Baltimore and London, 1990.

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

Wiegers, K.E. (1996). *Creating a Software Engineering Culture*. New-York : Dorset House Publishing.

Wolff, S. (1999), La place de la mesure au sein des principes fondamentaux du génie logiciel , in Département d'informatique, Montréal, Université du Québec à Montréal, 1999, pp. 131 .