Software Engineering Principles: 
A Survey and an Analysis

Normand Séguin, Université du Québec à Montréal
Alain Abran, École de technologie Supérieure
Robert Dupuis, Université du Québec à Montréal

Abstract: This study presents a survey and an analysis of the literature on software engineering principles. The literature survey, covering a period of thirty years, has come up with 14 different papers and books, which have proposed a total of 313 distinct principles for software engineering. Our analysis of these works is carried out based on a rigorous definition of the term ‘principle’ and on the identification a set of explicit criteria to assess whether or not any of the proposed principles qualify as fundamental principles of software engineering. The analytical approach makes it possible to arrive at a set of 24 candidate software engineering principles which all meet the required criteria.

Keywords: software engineering principles, fundamentals, laws, rules, concepts

1. Introduction

Research on the fundamental principles of software engineering is of interest not only to teachers with a desire to better define the curriculum of the discipline, but also for organizations developing standards for it. Standardization bodies do, in fact, have a mandate to develop and maintain a corpus of standards of software engineering: for instance, Moore (2006) stresses that this corpus currently constitutes a formalization of best practices. However, it has been developed in an ad hoc way during recent decades, and suffers from a lack of integration and coherence. As well, there is some overlapping. In order to remedy this situation, standards organizations need a better grasp of the foundations of the software engineering discipline.

Moore (2006) noted that, in the mature disciplines of engineering, it is possible to establish links between the standards and the engineering principles that constrain those standards. However, it is much less obvious how this can be achieved with regard to software engineering. First, software is an intangible product which is not constrained by the laws of physics. Second, software engineering is still emerging as an engineering discipline, having only recently been recognized as such.

Only a few researchers have attempted to identify the fundamentals of software engineering, and each has proposed a different set of principles. There has not yet been an attempt to analyze them, or to develop a basis from which to work progressively towards a consensus on
a core set of fundamental principles of software engineering. The key contributions of this paper are: better insight into the definition of a principle, a list of criteria to qualify a candidate principle as a fundamental software engineering principle, and a list of filtered software engineering principles which meet those criteria.

The research issue investigated in this paper is the following: among the 313 principles proposed in the literature between 1970 and 2003, which really represent the fundamentals of software engineering? There is no attempt in our work here to identify additional principles; instead, it filters and synthesizes the large number of principles in the literature survey. This paper reports on the structured approach adopted for the identification and consolidation of principles into a smaller set.

The paper is organized as follows. Section 2 presents a survey of related work on software engineering principles. Section 3 presents the research methodology used to analyze the proposed principles. Section 4 presents the conceptual framework proposed for this analysis. Section 5 presents the outcome of the application of individual criteria to each proposed principle, as well as the outcome of the application of overall criteria. Finally, section 7 presents a summary and suggestions for further research.

2. Related Work
2.1 Inventory of software engineering principles
Our literature survey has come up with 14 different papers and books in which a total of 313 distinct principles for software engineering have been proposed (Abran et al. 2003). Are these 313 proposed principles truly fundamental principles, and where do they come from? Table 1 chronologically lists the authors who put forward these proposed principles, together with the number put forward by each of them. From Table 1, it can be noted that Alan Davis (Davis 1995) has proposed the largest number by far (201 out of 313); however, it is inconceivable that the software engineering discipline rests on so many principles.
With the exception of Davis, most authors have not described their proposed principles in detail, nor have they indicated how to implement them in practice through the development life cycle. Davis gave a description and examples of each statement, which helps us to understand both their meaning and their scope. However, the consequences of not applying each principle are not stated. None of these authors has attempted to cover the whole domain of software engineering, nor have they documented what domain of software engineering is covered by the proposed principles.

### 2.2 Inconsistency of terminology

The literature survey reveals that the majority of the authors have used terms such as ‘principles’, ‘concepts’, ‘laws’, and ‘techniques’ in an imprecise way (Abran et al. 2003). For several of them, these terms are considered to be synonymous. Of course, this inconsistency in terminology leads to misunderstandings and makes arriving at a consensus difficult. The absence of precise definitions of the terms used, and consequently the lack of consensus within the software engineering community on a single definition of the term ‘principle’, is a source of confusion for researchers, as well as in the marketplace in general. This has led to different ways of formulating the principles proposed in the literature, such as:

1. a description of a concept rather than a principle;
2. a descriptive proposal; and
3. a prescriptive proposal guiding an action.
2.3 Methodology for identifying principles

Most authors did not document how they came up with their list of proposed principles. Therefore, we must assume that most are based on expert opinion, rather than on well-founded theories or on controlled and documented experimentation. Only Bourque et al. (2002) present an explicit description of the methodology designed specifically to develop, through a set of Delphi rounds with a group of experts, an initial consensus on a group of candidate fundamental principles, each identified based on the opinions of the Delphi participants and not on a study of previous work.

Furthermore, aside from describing a formal methodology, Bourque et al. proposed a definition of ‘principle’, something few authors have done. They also proposed criteria for identifying principles, some of which have been retained in this study. Even though 15 of the principles proposed by Bourque et al. are opinion-based, they have been validated by over 600 members of the software community through an online survey. Thus, compared to the other works cited, that of Bourque et al. differs on methodological grounds.

Finally, only Bourque et al. (2002) and Boehm (1983) have explicitly stated criteria for identifying principles.

In short, the effort to date to identify the fundamental principles of software engineering has led to 313 proposed principles. Not only is this number probably too high for the discipline, no indication of how much of the domain of software engineering is covered by them has been provided. Furthermore, the majority of the many lists of principles are simply based on the opinions of their respective authors.

From a research perspective, this opinion-based approach is a fairly weak means of identifying principles and developing a consensus on a core set. This could in part explain the hundreds of proposed principles, as well as the current lack of initiatives designed to synthesize the work done to date on the fundamentals of software engineering.

Addressing the issue first requires clarification of the term ‘principle’ itself, followed by identification of criteria against which to test the 313 proposed principles.
3. Research Methodology

To analyze all the proposed principles, the following research methodology, illustrated in Figure 1 (Seguin 2006), was adopted.

4. Conceptual Framework

For the development of criteria to analyze the pool of 313 proposed principles, six sets of concepts have been examined. These are listed in Figure 2 as the elements of the conceptual framework for the analysis of the proposed principles. Each is described below in further detail.

4.1 Definition of terms

The first element is the definition of the terms ‘concept’, ‘law’, and ‘principles’. This is of primary importance for the rigor of the analysis and the identification of gaps in the related work.
What is a concept?
Encyclopaedia Universalis (2002) mentions that human beings essentially use two modes for representing knowledge. The first is concrete and concerns knowledge with a direct link to an object in the real world. The other is abstract and is represented by concepts. These are characterized by their abstract and universal nature, and by being relatively stable over time. Such a concept may be expressed differently, depending on the language, but the concept keeps the same meaning. They unify the representation of objects and ideas.

Thirteen useful definitions were found for the term ‘concept’. Since many of them share common aspects, we first present the one in the dictionary, Le Robert (2002): ‘A mental and general representation of an idea, object, or general notion’. Bunge (2003) adds: “a simple idea, unit of meaning, building block of a proposition. Every concept can be symbolised by a term.”

The concept, as represented by a term, is the basic component of any proposition. A proposition is defined as ‘a statement which expresses a relation between two or more terms’ (Le Robert 2002). The concept would then be the basic unit of knowledge with which propositions, including principles and laws, would be formulated.

What is a principle?
The word ‘principle’ comes from the Latin term ‘principium’, meaning beginning or origin. Accordingly, the first meaning of this word in Le Robert refers to ‘origins, fundamental causes or constituent elements’ (Le Robert 2002). Principles are the foundation of any discipline, phenomenon, or functioning of an object.

A second meaning of the term is ‘a basic proposition, stated and not deduced’ (Le Robert 2002). A principle is basically a proposition which expresses a relation between two or more concepts (Le Robert 2002). A concept cannot by itself be considered as a principle. A proposition means, first, a principle that is not deduced. Furthermore, since a principle is intrinsically fundamental, it would be superfluous to speak about ‘fundamental principles’.

In addition to being a proposition, a principle also represents a rule, a law, or a general truth (Le Robert 2002, Bunge 2003). An interesting meaning of the term ‘rule’ is that it is an empirical rule adopted after tests have been conducted in practice. Such a rule serves as a
basis for action. This meaning of the term is highly relevant, since the principles of software engineering should constitute the source of an action and represent the basis of practice. Finally, a principle must be verifiable in terms of its consequences (Le Robert 2002), meaning that it must be possible to observe in its result, whether a principle has been applied or not. Also, the omission of a principle leads to error, which can be readily observed in the development of software (Litré 2004). The relationship between the terms discussed is shown in Figure 3.

Based on the sources previously discussed, the definition of the term ‘principle’ adopted in this study on the fundamental principles of software engineering is the following:

“A principle is a first and fundamental statement of the discipline formulated in a prescriptive manner in order to direct actions, and susceptible of being checked in terms of its consequences and by experiment.”

### 4.2 Identification criteria

The second element refers to the identification of criteria to verify whether or not a proposed principle is a fundamental principle. The criteria identified in the literature were divided into two groups:

- individual criteria to be applied individually to each principle; and
- overall criteria to be applied to the whole set of proposed principles.

Table 2 presents the seven criteria identified; that is, five individual criteria from Bourque et al. (2002) and two overall criteria from Bourque et al. (2002) and Boehm (1983).
Individual Criteria | Overall Criteria
--- | ---
1 - A principle is a proposition formulated in a directing way. | 1 - The principles should be independent (not deduced) (Boehm 1983).
2 - A principle should not be directly associated with, or rise from, a technology, a method, or a technique, nor be an activity of software engineering (adapted from Bourque et al. 2002). | 2 - A principle should not contradict another, known principle (Bourque et al. 2002).
3 - The principle should not state a compromise (or a proportioning) between two actions or concepts (adapted from Bourque et al. 2002). | 
4 - A principle of software engineering should include concepts connected to the discipline of engineering (Bourque et al. 2002). | 
5 - The formulation of a principle must make it possible to test it in practice, or to check it in terms of its consequences (Bourque et al. 2002). | 

Table 2: List of verification criteria for qualifying a proposed principle as a fundamental principle.

### 4.3 Categories of principles

The third element relates to categories of principles. Again, based on the literature review, three implicit categories of principles were identified: process, product, and resources (i.e. individuals). Throughout the analysis, the principles were classified according to these categories.

### 4.4 Engineering concepts

The fourth element is related to engineering concepts. Bourque et al. (2002) stress that the principles of software engineering should be associated with the principles of engineering themselves. Since certain concepts of engineering can be present in the formulation of some of the proposed principles, the generic concepts of engineering from Aslaksen (1996), Davis (1998), and Rodgers (1983) were identified, in order to detect them in the formulation of the proposed principles.
4.5 Software engineering concepts

The fifth element of the framework refers specifically to software engineering. In addition to defining software engineering, the SWEBOK Guide (Abran et al. 2004) provides the set of concepts of the discipline. The SWEBOK Guide has been widely recognized in the software engineering community as a guide to the domain’s knowledge areas. Identification of the concepts of software engineering makes it possible to recognize them in the formulation of the principles. Therefore, one of the verification criteria stipulates that a principle of software engineering must comprise at least one concept of software engineering. In this way, the SWEBOK Guide supports the application of this criterion.

4.6 Software engineering activities

The sixth element refers to identification of the fundamental activities of software engineering: a principle is not an activity in and of itself, but one or more activities can result from it. In order to verify whether or not the formulation of a principle represents an activity rather than a principle, ISO 12207 (ISO 1995) was used. This ISO standard explicitly identifies the processes and the basic activities of software engineering.

5. Application of the Individual Verification Criteria to the Inventory of 313 Proposed Principles

The five individual criteria were applied to the 313 proposed principles from the literature survey. Figure 4 presents the inputs and the initial output of the application of the individual criteria to the inventory of 313 proposed principles.

---

Figure 4: Application of the Individual Verification Criteria

The application of each of the five individual criteria gave the following results:
• Criterion 1, which requires that a principle be a directing principle, eliminated 52 of the proposed principles, each of which had been formulated primarily using only one word, representing a concept rather than a principle. For example, ‘abstraction’, ‘modularity’, and ‘metrics’ are not principles, but concepts. A further 37 were of a descriptive nature, and were also eliminated. Descriptive principles do not guide action.

• Criterion 2, which states that a principle must not be associated with a technology, a technique, or a software engineering activity eliminated 133 propositions. Most of these were activities explicitly mentioned in ISO 12207. A principle is not an activity per se, but activities can be derived from a principle. For example, “Test invalid inputs” is an activity included in the software process.

• Criterion 3, which states that a principle should not express a compromise (or a proportioning) between two actions or concepts, eliminated 3 propositions. For example, “use coupling and cohesion” and “use tools, but be realistic” do not meet the criterion.

• Criterion 4, which requires that a software engineering principle should at least include a concept from engineering or software engineering eliminated 126 propositions which did not contain any such concept explicitly. The software engineering concepts listed in the SWEBOK Guide were used. For example, “recursion or typing” is specifically related to computer science, not software engineering.

• Criterion 5, which states that the formulation of a principle must make it possible to test it in practice or to check its consequences. Many of the proposed principles did not refer to the consequences of being applied or not. Many of them (122) for which a dependent variable (that is, the consequences) could not be identified were eliminated. For example, it is not possible to check the consequences of the command, “Don’t believe everything you read.”

In summary, the application of individual criteria eliminated 273 of the proposed principles. However, since 18 of them had been eliminated solely on the basis of criterion 1 (non prescriptive formulation) while meeting the other 4 criteria, we attempted to reformulate these proposed principles in a prescriptive manner. It was possible to do this for 4 of them, however the others were strictly of a descriptive nature. As a result of applying the 5 individual criteria, 39 proposed principles were retained for the next phase of our study.
6. Application of the overall criteria

The next step consisted of applying the two overall criteria to the initial set of 39 principles selected in the first phase of the analysis, as follows:

1 - One principle should not contradict another principle in the set.
2 - The principles should be independent of one another (not deduced) (Boehm 1983).

In order to facilitate the application of the overall criteria, the 39 principles were classified according to the following three categories: process, product, and resources (individuals). The literature review revealed that some principles apply to the development process, while others are related to the product (software and intermediate products) and a few to the individuals carrying out the development process (Wiegers 1996) – see Table 3 for a summary. It should be noted that a principle can be classified in more than one category.

<table>
<thead>
<tr>
<th>Categories</th>
<th>Number of principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources (Individuals)</td>
<td>6</td>
</tr>
<tr>
<td>Product</td>
<td>7</td>
</tr>
<tr>
<td>Process</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 3: Breakdown of proposed principles by category.

The application of the first overall criterion did not result in the elimination of any of the proposed principles. Indeed, none of the 39 contradicted any other in the set.

The application of the second overall criterion eliminated 15 proposed principles in its first iteration, and so only 24 propositions met the two overall criteria, as well as the five individual criteria, of course.

![Diagram](image)

Figure 5: Application of the overall verification criteria

Table 4 presents the list of the 24 proposed principles from the survey that met all the individual and overall criteria.
1. Align incentives for developer and customer (Davis)
2. Apply and use quantitative measurements in decision making (Bourque)
3. Build software so that it needs a short user manual (Davis)
4. Build with and for reuse (Bourque)
5. Define software artifacts rigorously (Bourque)
6. Don’t overstrain your hardware (Davis)
7. Don’t try to retrofit quality (Davis)
8. Establish a software process that provides flexibility (Bourque)
9. Grow systems incrementally (Davis)
10. Implement a disciplined approach and improve it continuously (Bourque)
11. Invest in understanding the problem (Bourque)
12. Involve the customer (Royce)
13. Quality is the top priority; long-term productivity is a natural consequence of high quality (Wiegers)
14. Rotate (high performing) people through product assurance (Davis)
15. Since change is inherent to software, plan for it and manage it (Bourque)
16. Strive to have a peer, not a customer, find a defect (Wiegers)
17. Tailor cost estimation methods (Davis)
18. To improve design, study previous solutions to similar problems (Bourque)
19. Use better and fewer people (Boehm)
20. Use documentation standards (Davis)
21. Write programs for people first (Davis)
22. Know software engineering techniques before using development tools (Davis)
23. Select tests based on the likelihood that they will find faults (Davis)
24. Choose a programming language to ensure maintainability (Davis)

Table 4: List of the 24 proposed principles retained as candidate principles

In the second iteration, we focused our attention on the 15 proposed principles discarded by the 2nd overall criterion; that is, that a principle cannot be deduced. It was observed that these proposed principles could be deduced from other, more general proposed principles. However, some of them were more specific on the action to be taken; that is, the parent principle, which is of a more general nature, was not very specific.

For this reason, we added an additional iteration for these 15 proposed principles to determine whether or not they added precision to the related, more general one. Of these 15, 10 added precision with respect to guiding the action. From the point of view of the instantiation of these proposed principles in practice, these 10 principles would be more easily applicable than the parent principles.

25. Design for maintenance (Davis)
26. Determine requirements now (Davis)
27. Don’t write your own test plans (Davis)
28. Fix a requirements specification error now (Davis)
29. Give products to customers early (Davis)
30. Keep design under intellectual control (Davis)
31. Maintain clear accountability for results (Boehm)
32. Produce software in a stepwise fashion (Bourque)
33. Faced with unstructured code, rethink the module and redesign it from scratch. (Davis)

Table 5: Principles deduced from more general ones, but which add precision to the action to be taken
Table 6 presents an overview of which authors provided most of the proposed principles that met all seven verification criteria: the vast majority of those selected come from Davis (1995) and Bourque et al. (2002).

<table>
<thead>
<tr>
<th>Authors</th>
<th>Propositions</th>
<th>Retained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winston W. Royce (1970)</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Ross, Goodenough, and Irvine (1975)</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>H. D. Mills (1980)</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Many Lehman (1980)</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Barry W. Boehm (1983)</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Booch et Bryan (1984)</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Buschmann and Coll. (1996)</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Alan Davis (1995)</td>
<td>201</td>
<td>12</td>
</tr>
<tr>
<td>Karl Wiegers (1996)</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Anthony Wasserman (1996)</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Paul Taylor (2001)</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Bertrand Meyer (2001)</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Bourque et al. (2002)</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>Ghezzi, Jazayeri, and Mandrioli (2003)</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6: Breakdown by author(s) and years

7. Summary and Discussion

The literature survey on software engineering principles from 1970 identified a set of 313 proposed principles. While we cannot be certain that no others were published during that period, this list is certainly significant. However, the authors who put forward these candidate principles did not share a common approach for defining them, referring to them variously as laws, principles, and fundamental principles, and expressing them in several formats: definitions, descriptions, and prescriptions.

The conceptual framework designed for this study tackled some of the major methodological shortcomings of previous work. It defined the term “principle”, an essential step in the analysis of the proposed principles from the literature. It identified a list of seven criteria with which to analyze them, criteria that can be used to analyze any software engineering principles, as well as principles related to engineering in general. This feature sets our study apart from previous ones. Finally, our study has made it possible to eliminate from the list many proposed principles which were, in fact, standardized activities in the software life cycle. This highlights the importance of not confusing principles with activities, even when they are derived from activities.
We next applied our seven criteria to verify whether or not these proposed principles could be considered candidate fundamental principles of software engineering. This exercise enabled us to identify 24 such principles. We retained 10 additional principles that can be deduced from their respective parent principle, but which add precision to the action to be taken.

This shortened list of candidate principles can now be used to build consensus in the software engineering community. It may be that this list can be shortened yet again. Further work is also required to investigate the level of practice of these candidate principles, and how they can be operationalized throughout the software life cycle.

This analysis does not claim to provide the ultimate list of fundamental principles of software engineering. It may be that some principles of software engineering have yet to be identified in the literature. Should new ones be put forward, the criteria specified in the analysis could be used to verify whether or not they qualify as *bona fide* candidate fundamental principles. For instance, as we have noted, in terms of coverage of the software engineering field, the majority of the candidate principles belong to the process category, with only a few belonging to the product and resources categories. Moreover, there could well be gaps to be addressed.

Of the authors who have proposed principles, only a few have described or explained them. This analysis did not document all the principles retained. Consequently, a major addition to this study would be a description and examples of each candidate principle, as well as a description of the consequences of not applying that principle. Further work should be carried out to guide practitioners in applying the principles. Such documentation would help educators include adequate coverage of the principles in software engineering curricula.

Can these software engineering principles be used in open source software development or in Agile processes? Our research team has begun to work on finding answers to these questions (Séguin et al. 2006), but much more work is required.

In conclusion, a definitive list of software engineering principles would be highly useful for improving software engineering curricula and for training software engineers.

**References**


