J. Software Engineering & Applications, **2010, 3, 972-982**
doi:10.4236/jsea.2010.310114 Published Online October 2010 (http://www.SciRP.org/journal/jsea)

Scientific Research

Software Engineering Principles: Do They Meet Engineering Criteria?

Kenza Meridji, Alain Abran

École de Technologie Supérieure (ÉTS)—University of Québec, Montréal, Québec, Canada.
Email: kenza.meridji.1@ens.etsmtl.ca, alain.abran@etsmtl.ca

Received July 31st, 2010; revised August 25th, 2010; accepted August 31st, 2010.

ABSTRACT

As a discipline, software engineering is not as mature as other engineering disciplines, and it still lacks consensus on a well-recognized set of fundamental principles. A 2006 *analysis surveyed and analyzed* 308 *separate proposals for principles of software engineering, of which only thirty-four met the criteria to be recognized as such. This paper reports on a further analysis of these thirty-four candidate principles using two sets of engineering criteria derived from*: *A*) *the engineering categories of knowledge defined by Vincenti in his analysis of engineering foundations*; *and B*) *the joint IEEE and ACM software engineering curriculum. The outcome of this analysis is a proposed set of nine software engineering principles that conform to engineering criteria*.

Keywords*: *Engineering Criteria, Software Engineering Principles, Vincenti, Engineering Verification Criteria

1. Introduction

Software engineering is defined by the IEEE as:

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

2) The study of approaches as in 1" [1].

The intended goals of software engineering are different from those of computer science. In software engineering, artifacts are designed, produced and put into operation, while in computer science the theoretical foundations of information and computation are the object of study. Computer science deals with the investigation and analysis of algorithms and their related problems, in order to enable the computer perform the task [2].

Computer science is the discipline that underlies software engineering, and it is to be expected that the principles for computer science will be different from the principles of software engineering. For example, a principle proposed in computer science, "Use coupling and cohesion" [3], deals with the underlying science, while software engineering principles are more general, like "Apply and use quantitative measurements in decision making" [3].

Of course, software engineering is still an emerging engineering discipline and is not yet as mature as other traditional engineering disciplines such as mechanical and electrical engineering. Much of the research conducted to date in software engineering has focused on developing methods, techniques, and tools, and considerably less on exploring the engineering foundations of software engineering, including identifying the software engineering fundamental principles, or how to apply them in research and practice.

A significant amount of the work carried out to date on software engineering principles has been based on expert opinion, with a few exceptions in which defined research methodologies have been used, such as in [4] and [5], where Delphi rounds are applied to develop an initial consensus among a group of 12 experts on a set of candidate principles for software engineering.

In a 2006, literature survey on this topic, covering the previous 20 years [3], 308 separate proposals for candidate software engineering principles was identified. These were then analyzed against a set of criteria related to the specific concept of a 'principle', following which only 34 were recognized as bona fide 'candidate' fundamental principles (FPs) [3]. However, the research scope of that study did not, include within its research scope an analysis of these candidates from an engineering perspective.

In this paper, we perform that analysis. One of the challenges, of course, is to figure out what criteria should

be verified from an engineering perspective, since, in the traditional engineering literature, such criteria are not explicitly described. This paper documents the methodology to make that determination, as well as what we found when we applied them to the set of 34 candidate FPs.

The paper is organized as follows: Section 2 presents related work on software engineering principles. Section 3 presents the analysis methodology selected. Section 4 identifies the software engineering verification criteria. Section 5 describes the application of these criteria. Section 6 presents the analysis results. Section 7 points out the limitations of that work. Section 8 presents our conclusion.

2. Related Work on Software Engineering Principles

The expression "fundamental principle" is composed of two terms. According to the Cambridge University Dictionary, the term "fundamental" means "forming the base, from which everything else originates; more important than anything else", and "principle" means "a basic idea or rule that explains or controls how something happens or works".

From the literature on software engineering principles, the authors of [3] inventoried 308 principles that had been proposed by individuals (for instance [6-8]) or as part of a collaborative effort [9-12]. With the exception of [11] and [3], the authors involved proposed only nominative principles, without including either formal definitions or procedures for implementing them. To verify whether or not each of these was indeed a candidate 'fundamental' principle, in our sense of the terms, a two-step verification process was used in [3-11,13,14]:

1) Identification of seven verification criteria

Five criteria applicable to each proposed principle were derived from [4]:

- 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 technique, 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.

Two additional criteria were identified as applicable across the full set of proposed principles:

- The principles should be independent, e.g. not deduced [6];
- A principle should not contradict another known principle [4].

2) Verification of each of the proposed 308 principles surveyed against these criteria

In [3] it is reported that only 34 of the 308 proposals met the full set of criteria to be recognized as candidate FPs. Table 1 lists them, in alphabetical order [3].

In their paper "Fundamental Principles of Software Engineering–A Journey" [4], the authors identified a set of fundamental principles through a well documented research methodology. They defined the relationships between principles, standards and implemented best practices as illustrated in **Figure 1**.

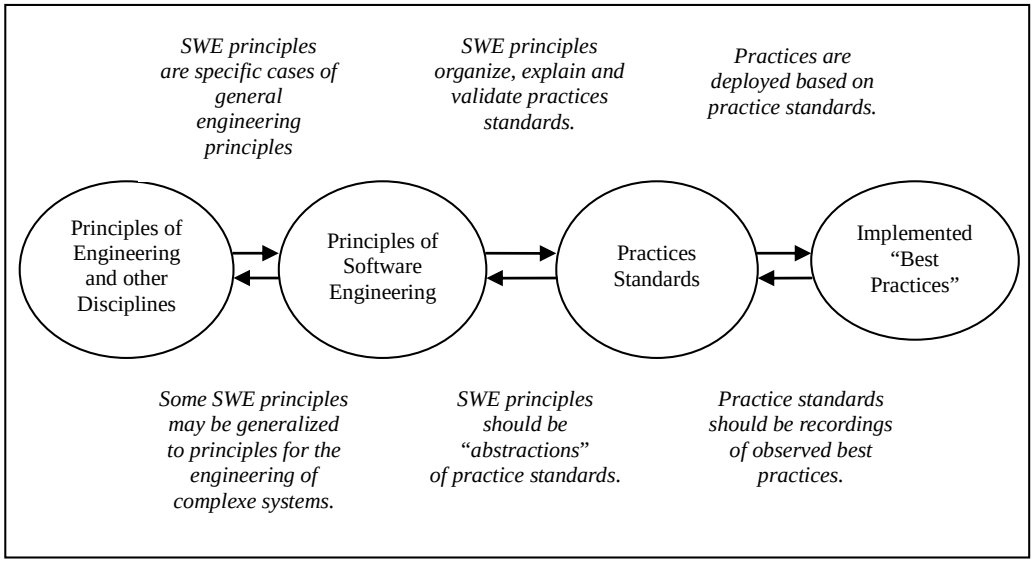

Figure 1. Relationships between principles, standards and practices [4].

This figure illustrates the relationships sought among principles standards and practices. It is believed that a body of fundamental principles has been recorded for some branches of engineering, e.g. [15]. Most of the engineering branches have a history far longer than that of software engineering and software engineering principles (SWE principles in **Figure 1**) would, in general, be regarded as specializations of these principles. The software engineering principles would play the role of organizing, motivating, explaining, validating the practice standards and implemented practices should be based on those practice standards [4].

Working from the specific toward the general, practice standards would be recordings and idealizations of observed and validated "best practices". The software engineering principles would be abstractions of the practice standards. Furthermore, software engineering principles might be candidates for generalization to the status of general engineering principles, particularly when complexity is a concern [4].

3. Analysis Methodology

The scope of the criteria used in [3] was limited to the concept of 'principles', and did not include the specific features of the engineering concepts themselves. This is the focus of the work reported here: the list of 34 candidate principles in **Table 1** constitutes the input to the analysis process required to verify whether or not they conform to engineering criteria. This will make it possible to narrow the number of principles. More specifically, the research issue addressed in this paper is, which of these 34 candidate FPs conform to engineering criteria?

Of course, engineering criteria are required for this verification and must be available, but no related work could be identified. So, the first challenge was to determine verification criteria from an engineering perspective.

To tackle this issue, it was necessary to study the epistemology of engineering. For that purpose, two sources, Vincenti, the author of the book, What Engineers Know and How They Know It [15], and the joint IEEE-ACM software engineering curriculum [16] were selected:

- Vincenti has identified a number of engineering knowledge types as key to the engineering disciplines and from which engineering criteria can be derived;
- The IEEE and the ACM have documented a set of topics within their joint software engineering curriculum from which engineering criteria can be derived.

The approach designed for identifying relevant criteria and applying them to the set of 34 candidate FPs consists of three phases—see **Figure 2**.

Phase 1: Identification of two sets of verification criteria

This phase consists of the identification of criteria which would be relevant to any engineering discipline. Such criteria could have been taken either as is, when expressly identified and defined, or derived, when documented only implicitly. The inputs to this phase are the two sources of information identified from the related work. The outputs of this phase are the two sets of criteria derived from Vincenti and from the IEEE-ACM joint software engineering curriculum. The criteria identification phase based on Vincenti is summarized in **Figure 3**, which shows its inputs and outputs.

The criteria identification phase based on the IEEE-ACM criteria is summarized in **Figure 4**, which shows its inputs and outputs. This phase is presented in greater detail in Section 4.

Phase 2: Verification execution:

The 34 candidate FPs will be taken as inputs in the second phase and analyzed next with respect to the two sets of engineering criteria identified in Phase 1.

The output will be the FPs that have at least one direct mapping, and those that have only an indirect mapping to either Vincenti or to the IEEE-ACM engineering criteria. This phase is illustrated in **Figure 5** and is presented in greater detail in Section 5.

Phase 3: Analysis and selection:

In Phase 3, the analysis across each set of engineering criteria is performed. This phase identifies the candidate FPs that meet engineering criteria from both sets of criteria and those that do not. For instance, the candidate FPs that meet only the Vincenti criteria [15] and the candidate FPs that only meet the IEEE-ACM criteria [14] are then be analyzed to check whether or not they can be identified from the FP that are recognized as engineering FPs. This phase is described in greater detail in Section 6.

4. Phase 1: Identification of Engineering Criteria

4.1. Vincenti

Vincenti [17] studied the epistemology of engineering based on the historical analysis of five case studies in aeronautical engineering covering a roughly fifty-year period and proposed a taxonomy of engineering knowledge. He identified different types of engineering knowledge and classified them into six categories:

1) Fundamental design concepts,
2) Criteria and specifications,
3) Theoretical tools,
4) Quantitative data,
5) Practical considerations, and

Table 1. Inventory of Candidate FPs [3].

No	Proposals–in alphabetical order
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
5	Define software artifacts rigorously
6	Design for maintenance
7	Determine requirements now
8	Don't overstrain your hardware
9	Don't try to retrofit quality
10	Don't write your own test plans
11	Establish a software process that provides flexibility
12	Fix requirement specification errors now
13	Give product to customers early
14	Grow systems incrementally
3	Implement a disciplined approach and improve it continuously
16	Invest in understanding the problem
17	Involve the customer
18	Keep design under intellectual control
19	Maintain clear accountability for results
20	Produce software in a stepwise fashion
21	Quality is the top priority; long-term productivity is a natural consequence of high quality
22	Rotate (top performing) people through product assurance
23	Since change is inherent to software, plan for it and manage it
24	Since tradeoffs are inherent to software engineering, make them explicit and document them
25	Strive to have a peer find a defect, rather than a customer
26	Tailor cost estimation methods
27	To improve design, study previous solutions to similar problems
28	Use better and fewer people
29	Use documentation standards
30	Write programs for people first
31	Know software engineering techniques before using development tools
32	Select tests based on the likelihood that they will find faults
33	Choose a programming language to ensure maintainability
34	F aced with unstructured code, rethink the module and redesign it from scratch

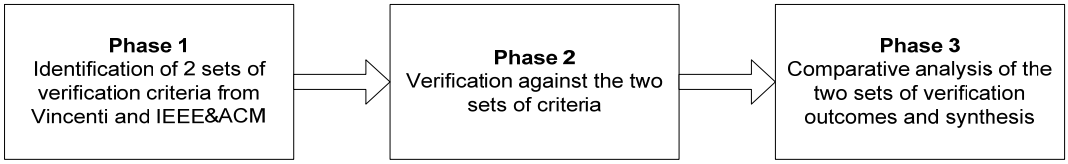

Figure 2. The three-phase verification process.

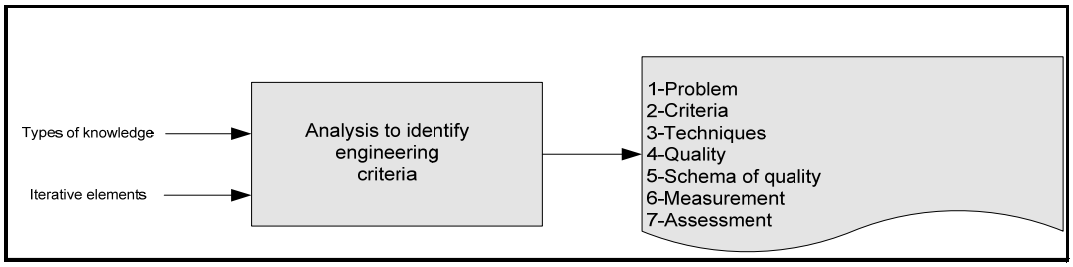

Figure 3. Identification of Vincenti engineering criteria.

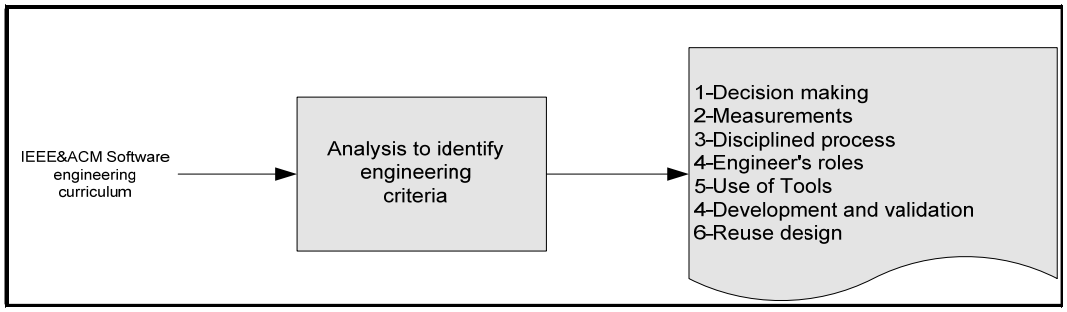

Figure 4. Identification of the IEEE-ACM engineering criteria.

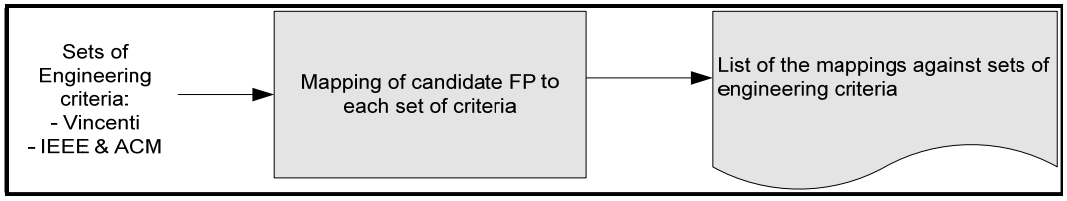

Figure 5. Phase 2: Process of verification against sets of engineering criteria.

6) Design instrumentalities.

According to Vincenti, this classification is not specific to aeronautical engineering, but can be transferred to other engineering domains. For instance, a detailed analysis of engineering knowledge types was used in [18] to analyze the content of the Software Quality Knowledge Area of the Guide to the Software Engineering Body of Knowledge–SWEBOK [15].

Vincenti has distinguished seven elements for engineering, which he refers to as "interactive elements", and which he selected prior to categories of engineering knowledge types. These elements show the epistemological structure of the engineering learning process based on the analysis of the five aeronautical case studies. These seven elements represent, in Vincenti's opinion, a necessary set of different elements that interact with each other for the completion of an engineering activity. These seven interactive elements are referred to here as the Vincenti engineering criteria, and are listed in **Table 2**. The abbreviations we have selected to represent each of these criteria are listed in the right-hand column of **Table 2**.

4.2. IEEE and ACM Joint Curriculum

The IEEE Computer Society (IEEE-CS) and the Association for Computing Machinery joined forces to develop a joint set of computer curricula, including one on software engineering. More specifically, chapter 2 of the joint software engineering curriculum lists the characteristics of an engineering discipline (see **Table 3**). These

Table 2. Engineering criteria identified in Vincenti.

ID.	Vincenti Engineering Criteria	Abbreviation
1	Recognition of a problem	Problem
2	Identification of concepts and criteria	Criteria
3	Development of instruments and techniques	Techniques
4	Growth and refinement of opinions regarding desirable qualities	Quality
5	Combination of partial results from 2, 3 and 4 into practical schema for research	Testing
6	Measurement of characteristics	Measurement
7	Assessment of results and data	Assessment

Table 3. Identification of IEEE & ACM engineering criteria.

ID.	Engineering Criteria Identified	Abbreviation
1	Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision point that is appropriate for the current task in the current context. Appropriateness can be judged by tradeoff analysis, which balances costs against benefits.	Decision making
2	Engineers measure things, and, when appropriate, work quantitatively; they calibrate and validate their measurements; and they use approximations based on experience and empirical data.	Measurements
3	Engineers emphasize the use of a disciplined process when creating a design and can operate effectively as part of a team in doing so.	Disciplined process
4	Engineers can have multiple roles: research, development, design, production, testing, construction, operations, management, and others, such as sales, consulting, and teaching.	Engineer's roles
5	Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering.	Use of Tools
6	Engineers, via their professional societies, advance by the development and validation of principles, standards, and best practices.	Development and validation
7	Engineers reuse designs and design artifacts.	Reuse design

characteristics are adopted here as engineering verification criteria. The abbreviations we have selected to represent each of these criteria are listed in the right-hand column of **Table 3**.

5. Phase 2: Verification against the Two Sets of Criteria

The set of 34 candidate FPs is next mapped to the two sets of engineering criteria: each candidate FP is taken as input and analyzed using each of Vincenti's seven criteria and, again, each of the seven IEEE-ACM software engineering criteria.

The output of the mapping to Vincenti's engineering criteria is presented in Appendix A-1, where the letter D represents a direct mapping, and the letter I an indirect one. For instance:

- Candidate FP #2 (Apply and use quantitative measurements in decision making) maps directly to Vincenti's criterion #6 and indirectly to Vincenti's criterion #4.
- Candidate FP #31 (Know software engineering techniques before using development tools) has only an indirect mapping to criterion #3 and to criterion #7 (Assessment).
- Finally, there are candidate FPs with no mapping to any engineering criteria: for instance, candidate FP #13 (Give product to customers early).

This first verification against the Vincenti criteria leads to the following results (see Appendix A-1):

- 12 candidate FPs have at least one direct mapping to a Vincenti engineering criterion;
- 21 candidate FPs have only indirect mappings to Vincenti engineering criteria;
- 1 candidate FP has no direct or indirect mapping to any Vincenti engineering criteria.

The second verification against the seven IEEE & ACM engineering criteria is presented in Appendix A-2.

For instance:

- Candidate FP #2 (Apply and use quantitative measurements…) has a direct mapping to criteria #1 (Decision making) and #2 (Measurements). Candidate FP #16 (Invest in the understanding of the problem) is mapped indirectly to criteria #1 (Decision making) and #3 (Disciplined process).
- Candidate FP #4 (Build with and for reuse) is mapped directly and indirectly to criteria #7 (Reuse) and #3 (Disciplined process).
- Finally, candidate FP #13 (Give products to customers early) is not related to any engineering criteria.

This second verification against the IEEE and ACM criteria leads to the following results (see Appendix A-2):

- 15 candidate FPs have at least one direct mapping to an IEEE-ACM engineering criterion;
- 16 candidate FPs have only indirect mappings to an IEEE-ACM engineering criterion;
- 3 candidate FPs have neither direct nor indirect mappings to any IEEE-ACM engineering criteria.

6. Phase 3: Analysis and Consolidation Using Both Sets of Criteria

6.1. Analysis across Each Set of Engineering Criteria

The candidate FPs with a direct mapping to either the Vincenti or IEEE-ACM criteria are listed in **Table 4**. From a comparison of the two columns in this table, the candidate FPs with direct mappings can be grouped into three sets:

1) Candidate FPs with a Vincenti mapping similar to the IEEE-ACM mapping;

2) Candidate FPs with a Vincenti mapping with no equivalent IEEE-ACM mapping;

3) Candidate FPs with an IEEE-ACM mapping with no equivalent Vincenti mapping.

Table 4. Candidate FPs that directly meet criteria from either set.

#	Vincenti Mapping	#	IEEE-ACM Mapping
2	Apply and use quantitative measurements in decision making	2	Apply and use quantitative measurements in decision making
4	Build with and for reuse	4	Build with and for reuse
		5	Define software artifact rigorously
		6	Design for maintenance
7	Determine requirements now		
9	Don't try to retrofit quality		
		10	Don't write your own test plans
11	Establish a software process that provides flexibility	12	Fix requirements specification error now
14	Grow systems incrementally		
15	Implement a disciplined approach and improve it continuously	15	Implement a disciplined approach and improve it continuously
16	Invest in understanding the problem		
		18	Keep design under intellectual control
21	Quality is the top priority; long term productivity is a natural consequence of high quality	21	Quality is the top priority; long term productivity is a natural consequence of high quality
		22	Rotate (high performing) people through product assurance
23	Since change is inherent to software, plan for it and manage it		
24	Since tradeoffs are inherent to software engineering, make them explicit and document them	24	Since tradeoffs are inherent to software engineering, make them explicit and document them
		25	Strive to have a peer, find a defect rather than a customer
		26	Tailor cost estimation methods
27	To improve design, study previous solutions to similar problems	27	To improve design, study previous solutions to similar problems
		31	Know software engineering techniques before using development tools

Set A:

From **Table 4**, we can see that six candidate FPs (#2, #4, #15, #21, #24, #27) are present in both columns (the highlighted ones) and therefore satisfy at least one engineering criterion in each set of criteria (Vincenti and IEEE-ACM): these six could reasonably be considered as FPs that conform to engineering criteria.

Set B:

From **Table 4**, we note that there are 6 other candidate FPs that meet the Vincenti criteria, but no IEEE-ACM criteria, and 9 candidate FPs that meet the IEEE-ACM criteria, but no Vincenti criteria. Can these still be considered as FPs, or are they merely instances of more fundamental principles?

To answer this question, we could reasonably argue from the Vincenti subset that:

- Candidate FP #7 (Determine requirements now) can be deduced from candidate FP #16 (Invest in understanding the problem);
- Candidate FP #9 (Don't try to retrofit quality) can be deduced from candidate FP #21 (Quality is the top priority; long-term productivity is a natural consequence of high quality);
- Candidate FP #11 (Establish a software process that provides flexibility) can be deduced from FP #9 (Don't try to retrofit quality).

This would eliminate candidates FPs #7, #9, and #11 from the list of FPs, since they represent specific instantiations of more general FPs, while principles #16, #14, and #23 would be retained on the list of FPs.

Set C:

From **Table 4**, there remain 9 candidate FPs without a corresponding direct mapping to the Vincenti criteria. We could reasonably argue that these 9 can be deduced from those with direct Vincenti mappings: for instance, FP #18 (Keep design under intellectual control) and FP #31 (Know software engineering techniques before using development tools) can be deduced from FP #15 (Implement a disciplined approach and improve it continuously).

This would eliminate candidate FPs #5, #6, #10, #12, #18, #22, #25, #26, and #31 from the list of FPs, since they represent specific instantiations of more general FPs, while retaining principle #15.

In summary, this analysis has allowed us to refine the list of 34 candidate principles to 9 software engineering principles based on engineering criteria. This analysis has also eliminated the overlap between the various principles; as a consequence, a subset of only 9 (see **Table 5**) from the list of 34 candidates identified in Seguin 2006 are recognized as principles that conform to engineering criteria, the remaining 25 being specific instantiations of those 9. In **Table 5**, these software engineering FPs are sequenced from 1 to 9, along with their original sequence number (right-hand column) assigned when the initial list of 34 candidates was compiled.

6.2. Identification of a Hierarchy

Table 6 presents next the outcome of our analysis of the 25 remaining candidate FPs as instantiations of the 9 principles in **Table 5** that conform to engineering criteria.

7. Work Limitations

The initial list of 34 candidates taken as input for this research is not necessarily exhaustive: to summarize, these 34 candidates have been refined from 304 proposed principles identified in the literature over a period of 20 years, up to 2006 [19]. The methodology used engineering

Table 5. List of 9 software engineering principles.

#	Vincenti, IEEE-ACM mapping	
1	Apply and use quantitative measurements in decision making	2
2	Build with and for reuse	4
3	Grow systems incrementally	14
4	Implement a disciplined approach and improve it continuously	15
5	Invest in the understanding of the problem	16
6	Quality is the top priority; long term productivity is a natural consequence of high quality	21
7	Since change is inherent to software, plan for it and manage it	23
8	Since tradeoffs are inherent to software engineering, make them explicit and document it	24
9	To improve design, study previous solutions to similar problems	27

Table 6. Hierarchy of candidate principles for software engineering.

#	Direct mapping to Vincenti criteria		Derived instantiation (= Indirect mapping)
2	Apply and use quantitative measurements in decision making	26	Tailor cost estimation methods
		8	Don't overstrain your hardware
4	Build with and for reuse		
14	Grow systems incrementally	5	Define software artifacts rigorously
		20	Produce software in a stepwise fashion
15	Implement a disciplined approach and improve it continuously	1	Align incentives for developer and customer
		10	Don't write your own test plans
		17	Involve the customer
		18	Keep design under intellectual control
		20	Produce software in a stepwise fashion
		31	Know software engineering's techniques before using development tools
		19	Maintain clear accountability for results
		29	Use documentation standards
		10	Don't write your own test plans
16	Invest in understanding the problem	7	Determine requirements now
		12	Fix requirements specification error now
		17	Involve the customer
21	Quality is the top priority; long term productivity is a natural consequence of high quality	9	Don't try to retrofit quality
		22	Rotate (high performing people through product assurance
		25	Strive to have a peer find a defect rather than a customer,
		30	Write programs for people first
		3	Build software so that it needs a short user manual
		11	Establish a software process that provides flexibility
		28	Use better and fewer people
23	Since change is inherent to software, plan for it and manage it	6	Design for maintenance
		33	Choose a programming language to ensure maintainability
		32	Select tests based on the likelihood that they will find faults
		34	In the face of unstructured code, rethink the module and redesign it from scratch.
24	Since tradeoffs are inherent to software engineering, make them explicit and document them		
27	To improve design, study previous solutions to similar problems		

criteria from Vincenti and the joint IEEE & ACM software engineering curriculum to identify the 9 candidate principles that conform to these engineering criteria; however, this list of criteria is not necessarily exhaustive, and more criteria could eventually be added.

Furthermore, most of the authors who proposed these principles did not provide operational descriptions of their proposals, and did not provide research experimentation for each principle identified.

8. Conclusions

Software engineering, as a discipline, is certainly not yet as mature as other engineering disciplines, and, while a number of authors have proposed over 300 distinct FPs, a consensus on a set of well-recognized FPs has been lacking. This research report has taken as input, or as its object of study, the set of 34 statements identified in [19] as candidate FPs of software engineering. This set has been analyzed from an engineering perspective using the engineering criteria identified by either Vincenti or the IEEE-ACM joint effort on developing a software engineering curriculum.

The 34 candidate FPs were divided into three categories: A) candidate FPs that are directly linked to engineering, B) candidate FPs that are indirectly linked to engineering, and C) candidate FPs with no specific link to engineering.

In the next step, the candidate FPs that were generic were distinguished from the more specific ones: this distinction was based on the type of mapping (direct or indirect). In the final step, candidate FPs from both lists were analyzed and compared. Our proposed reduced list of 9 software engineering principles now needs to be further discussed by the software engineering community.

Of course, this list depends on the methodology used, and is being proposed to the engineering community for discussion and scrutiny with the aim of improving it and developing a consensus over time.

There is no claim that this list is exhaustive or that it covers the whole software engineering discipline. Even though the inputs to this analysis were derived from an extensive literature survey, this does not guarantee that those authors have indeed provided full coverage of the software engineering discipline.

Similarly, the hierarchy proposed in **Table 6** is derived from the engineering criteria in our analytical approach. Further research should be carried out to verify the completeness of the criteria used.

REFERENCES

[1] IEEE Std 610.12, IEEE Standard Glossary of Software Engineering Terminology. Corrected Edition, February 1991.

[2] K. E. Wiegers, "Creating a Software Engineering Culture," Dorset House Publishing, New York, USA, 1996.

[3] N. Séguin, "Inventaire, Analyse et Consolidation des Principes Fondamentaux du Génie Logiciel," *École de technologie supérieure*, Université du Québec, Québec, Canada, 2006.

[4] P. Bourque, R. Dupuis, A. Abran, J. W. Moore, L. Tripp and S. Wolff, "Fundamental Principles of Software Engineering–A Journey," *Journal of Systems and Software*, Vol. 62, No. 1, 2002, pp. 59-70.

[5] Jabir and J. W. Moore, "A Search For Fundamental Principles of Software Engineering," Report of a Workshop Conducted at the Forum on Software Engineering Standards Issues, Computer Standards & Interfaces–In- ternational Journal on the Development and Application of Standards for Computers, Data Communications and Interfaces, Elsevier, Amsterdam, North Holland (the participants at this workshop names their group "Jabir"), Vol. 19, No. 2, 1998, pp. 155-160.

[6] B. W Boehm, "Seven Basic Principles of Software Engineering," *Journal of Systems and Software*, Vol. 3, No. 1, 1983, pp. 3-24.

[7] A. M. Davis, "201 Principles of Software Development," McGraw-Hill, New York, USA, 1995.

[8] K. E. Wiegers, "Creating a Software Engineering Culture," Dorset House Publishing, New York, 1996.

[9] P. Bourque and R. Dupuis, "Fundamental Principles of Software Engineering," *Third International Symposium and Forum on Software Engineering Standards*, Walnut Creek, CA, USA, 1997.

[10] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern Oriented Software Architecture," John Wiley & Sons, England, 1996.

[11] C. Ghezzi, M. Jazayeri and D. Mandrioli, "Fundamentals of Software Engineering," Prentice Hall, New Jersey, 2003.

[12] P. Bourque and R. Dupuis, "Fundamental Principles of Software Engineering," 3*rd International Symposium and Forum on Software Engineering Standards*, Walnut Creek, CA, 1997.

[13] A. Abran, N. Seguin, P. Bourque and R. Dupuis, "The Search for Software Engineering Principles: An Overview of Results," *Conference on the Principles of Software Engineering*, Buenos Aires, Argentina, 2004.

[14] N. Séguin and A. Abran, "Inventaire des principes du génie logiciel," *Revue Génie Logiciel*, Numéro 80, Paris, France, 2007, pp. 45-51.

[15] W. G. Vincenti, "What Engineers Know and How They Know It—Analytical Studies from Aeronautical History," Johns Hopkins University, Baltimore, MD, USA, and London, UK, 1990.

[16] IEEE and ACM, "IEEE Computer Society and Association for Computing Machinery," Curriculum Guidelines for Undergraduate Degree Programs in Software Engi-

neering, A Volume of the Computing Curricula Series, 2004.

[17] A. Abran, J. W. Moore, P. Bourque and R. Dupuis, "Guide to the Software Engineering Body of Knowledge," 4th Edition, In: P. Bourque and R. Dupuis, Eds., *IEEE Computer Society*, Los Alamitos, CA, USA, 2005.

[18] A. Abran and K. Meridji, "Analysis of Software Engineering from an Engineering Perspective," *European Journal for the Informatics Professional*, Vol. 7, No. 1, February 2006, pp. 46-52.

[19] SO-TR-19759, "Software Engineering—Guide to the Software Engineering Body of Knowledge (SWEBOK)," *International Organization for Standardization*, Switzerland, 2005.

