ELSEVIER

Computer Standards & Interfaces 19 (1998) 155–160

COMPUTER STANDARDS & INTERFACES

A search for fundamental principles of software engineering [1]

Jabir [2], J.W. Moore [a,*]

[a] *7803 Whiterim Terrace, Potomac, MD 20854, USA*

Accepted 29 September 1997

Abstract

In the 50-year history of software development, various methodologies and techniques have been proposed to facilitate the development of software responsive to needs. Most have proved to be more specific to the then-current state of technology than has been understood at the lime. At this time. enough examples have accumulated that we can begin to perceive underlying principles that may be fundamental, hence enduring in applicability. This paper reports the results of a recent workshop, discussing the characteristics and criteria for identifying fundamental principles and the application of those criteria to eight candidates. Recommendations for continuing work are provided. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Software engineering: Fundamental principles; Best practice

1. Context

Our interest in the identification of the fundamental principles of software engineering results from work in the development of software engineering practice standards. It is widely posited that practice standards should be based upon observation, recording and consensual validation of implemented 'best practices'. This strategy has resulted, though, in the development of a corpus of standards that are sometimes alleged to be isolated, unconnected and *dis*-integrated, because each standard performs a local optimization of a single observed practice. It is hoped that the identification of a set of fundamental principles would provide a broad and rich framework for establishing relationships among groups of prac-

* Corresponding author. Tel.: + 1 703 8837396; fax: + 1 703 8835432; e-mail: moorej@acm.org

[1] Report of a workshop conducted at the Forum on Software Engineering Standards Issues, Montréal, Quebec. Canada. 2 l-25 October 1996.

[2] Jabir is the collective name adopted by a group of individuals engaged in a search for the fundamental principles of software engineering. Our namesake was an eighth century Arabian alchemist who added sulfur and mercury to the then accepted list of elements-earth. fire, air and water [1]. The selected name suggests that our search has made progress toward truth but is far from completed. The following individuals have contributed to Jabir through personal participation or correspondence. Collective reports published under the name of Jabir do not necessarily represent their individual opinions. Affiliations are shown only for the purpose of Identification: (1) Alain Abran, Université du Québec à Montréal; (2) Pierre Bourque, Université du Québec à Mont&l; (3) Business Planning Group. IEEE Computer Society Software Engineering Standards Committee; (4) Robert Dupuis, Université du Québec à Montréal; (5) Duane Hybertson, The MITRE Corporation; (6) Jean-Philippe Jaquet. Université du Québec à Montréal; (7) Andreas Köller, Competence Center Informatik; (8) Ed Lowry: (9) James W. Moore, The MITRE Corporation; (10) Leonard Tripp. Boeing Commercial Airplane.

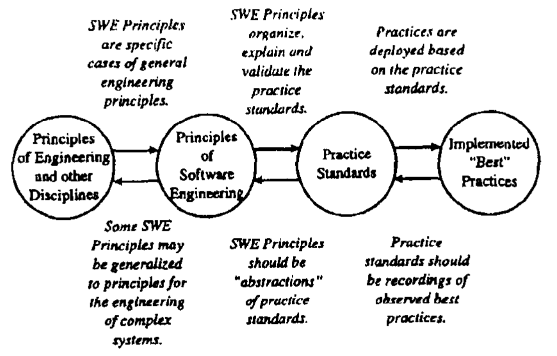
SWE Principles are specific cases of general engineering principles.
SWE Principles organize, explain and validate the practice standards.
Practices are deployed based on the practice standards.
Principles of Engineering and other Disciplines
Principles of Software Engineering
Practice Standards
Implemented "Best" Practices
Some SWE Principles may be generalized to principles for the engineering of complex systems.
SWE Principles should be "abstractions" of practice standards.
Practice standards should be recordings of observed best practices.

Fig. 1. Relationship of principles and practices.

tice standards, The 1996 Montréal workshop initiated a long-term, if episodic, inquiry to articulate such principles.

2. Relationship of principles, standards and practices

Fig. 1 illustrates the sought relationships among principles, standards and practices. It is believed that a body of fundamental principles for engineering and some other disciplines already exists and is articulated. (Most of the relevant disciplines have a history far longer than software engineering.) Software engineering principles would, in the general case, be regarded as specializations of the principles of these more fundamental disciplines. The software engineering principles would play the role of organizing, motivating, explaining and validating the practice standards. Implemented practices would be based on those practice standards. 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.

3. Results of the Montreal workshop

A workshop was organized at the 1996 Forum on Software Engineering Standards Issues [2] with the objective of identifying some of these fundamental principles. Position papers for the Montreal workshop were invited in the context presented above. Three days of discussion resulted in:

- Some-observations on the nature of fundamental principles,
- Some criteria for identifying and evaluating candidate principles,
- Some worked examples and counterexamples,
- Some recommendations for future work.

4. The nature of software engineering

To bound the scope of software engineering, we resorted to reference to authority. Two definitions seemed particularly useful: (1) "That form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems" [3]. (2) "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software" [4].

It was accepted at the outset that any particular articulation of fundamental principles is always imperfect. In fact, we established a metaphorical equation:

$$\begin{aligned}\text{Conventional\_Wisdom} &= \text{Folklore}\\ &+ \text{Fundamental-Principles}\end{aligned}$$

It was agreed that practice standards, at any point in time, are a recording of conventional wisdom. Fundamental principles are discovered by stripping away the folklore component of conventional wisdom.

5. Problems in discovering the fundamental principles of software engineering

It is not easy to find, isolate and articulate the relevant principles although some attempts have been made, e.g., [5–8]. Software engineering lacks a well-recorded, well-organized, accepted body of knowledge [9]. Even in the more mature disciplines where the fundamental principles are putatively known, the knowledge is often tacit. Validation of a suspected principle is difficult because software experiments are generally difficult to perform and because the identification of best practice is generally based on anecdotally based consensus judgment rather than experimentation.

Finally, immature disciplines (and software engineering is just one example) manifest an important qualitative difference from the more mature ones. They emphasize the problem-solving process (as if every problem were novel) rather than categorizing problems into well-understood bins and applying 'cookbook' solutions. This difference in focus is a fundamental and substantial barrier to the application of general engineering principles to software engineering. This barrier is expected to decrease over lime as software engineering matures.

6. Characteristics of fundamental principles

Despite the difficulties, we identified some characteristics of the desired fundamental principles, partly by a priori reasoning and partly through our experience in examining candidate principles.

• Any particular statement of a fundamental principle is imperfect (but this should not be regarded as an embarrassment).

• At any time, some fundamental principles are tacit.

• Software engineering's fundamental principles are derived by 'multiple inheritance' from other disciplines by adopting them as is, by specializing them, restating them, combining them, etc.

• We are willing to believe (for working purposes) that software engineering may have some unique principles. (This may be regarded as a consequence of the observation that the previously listed three characteristics may profoundly obscure the perception of the relationship to more general principles.)

In our examination of candidate principles, we developed some criteria (possibly better regarded as heuristics or meta-principles) for the recognition of fundamental principles (FP):

• FPs are less specific than methodologies and techniques, i.e., specific methodologies and techniques may be selected, ***within a particular technological context***, to accomplish the intent of FPs.

• FPs are more enduring than methodologies and techniques, i.e., FPs should be phrased in a way that will stand the 'test of time' rather than in the context of current technology.

• FPs are typically discovered or abstracted from practice and should have some correspondence with 'best' practice.

• Software engineering FPs should nor contradict more general FPs; but, there may be trade-offs in the application of FPs.

• An FP should not conceal a trade-off. By that we mean that an FP should not attempt to prioritize or select among various qualities of a solution; the engineering process should do that. FPs should identify or explain the importance of the various qualities among which the engineering process will make trades.

• An FP should be precise enough to be capable of support or contradiction.

• An FP should relate IO one or more underlying concepts. (See Section 7.)

7. Underlying concepts versus fundamental principles

In our search for fundamental principles, we discovered that we often confused them with something we eventually characterized as 'underlying concepts'. The following table contrasts the two:

Underlying concepts	Fundamental principles
Scientific	Engineering
Descriptive	Prescriptive
Validated through experiment	Validated through rigorous (but not necessarily experimental) assessment of practice
Judged on the basis of correctness	Judged on the basis of usability, relevance, significance, usefulness

Underlying concepts are to be regarded as scientific statements. They must be capable of validation by experiment and are judged on the basis of their correctness when subjected to experiment. On the other hand, fundamental principles are to be regarded as engineering statements which prescribe constraints on solutions to problems or constraints on the process of developing solutions. They should be rigorously evaluated, but in practice rather than in the laboratory, and judged by whether they provide useful and substantial contributions to the successful solution of real problems of significant size and scope. In general, we would expect fundamental engineering principles to be strongly related to underlying scientific **concepts.**

8. Candidate fundamental principles

This section of the paper describes the workshop's consideration of several candidate principles submitted from various sources. In each case, the candidate is stated followed by a discussion of its fit to the criteria described previously. In most cases, the candidate is restated or replaced with better candidates.

8.1. *Candidate 1*

8.1.1. First statement

Design quality comes more from knowledge of previous solutions than the specific requirements of the problem.

8.1.2. *Discussion*

The statement is descriptive rather than prescriptive suggesting that, in its current form, it is a candidate to be an underlying concept rather than a fundamental principle. It might be true, suggesting that stated requirements are inevitably incomplete because important requirements are often so obvious to the acquirer that they go unstated, even unperceived.

An important problem with the statement is that it hides a trade-off. It says that one approach is more effective than another without explaining how the two should be traded.

The phrasing of this statement suggests that it is actually more general than software engineering, possibly applying to the whole of engineering or, even more generally, to design (including, for example, graphics arts). Indeed, one participant suggested that empirical validation can be found in the field of architecture (of buildings). For our purposes, we ignore the possibility of over-generality and, instead, deal with the problems of failing to be prescriptive and of hiding a trade-off.

8.1.3. Restatement

To improve design, study previous solutions to similar problems.

8.2. *Candidate 2*

8.2. 1. First *Statement*

Control complexity with multiple perspectives and multiple levels of abstraction.

8.2.2. *Discussion*

This candidate is stated prescriptively and some techniques that can be traced to the statement have been empirically validated. In fact, various phrasings of this principle, usually more specific, have been made for the past twenty-five years or so. This is an example of a principle that is possibly specific to software engineering, or other disciplines with problems of comparable complexity. Our conclusion is that it is a fundamental principle and that no restatement is needed.

8.3. *Candidate 3*

8.3. I. First statement

Make sure that you have valid requirements before developing a solution.

8.3.2. *Discussion*

This candidate appears in many articles and textbooks. In fact, some claim that violation of this principle is the leading cause of failure in software development programs. In fact, though, any practitioner knows that the real requirements of the job change during the development and that failure to recognize this reality is also a leading cause of failure. So, there is some element of truth in the candidate but it is too imperfect to be useful; in some circumstances, application of the candidate might actually be harmful Two restatements seem appropriate.

8.3.3. *Restatements*

Pattern the solution after prior solutions to similar problems. (Thus improving the chances of capturing unknown requirements.)

"Software engineering, as part of a large system design process, must recognize the ill-defined and fluid nature of software requirements and take appropriate steps to ensure that the resulting products faithfully meet the user's true needs" [3].

8.4. *Candidate 4*

8.4.1. First Statement

Realize that software entropy increases (similar to a principle in Ref. [6]).

8.4.2. *Discussion*

This is a descriptive statement hiding behind an imperative, hence more likely to be an underlying concept rather than a fundamental principle. It also appears to be software-specific but seems to rely on a metaphor with thermodynamics to provide validity. Several possible restatements might be considered.

8.4.3. *Restatements*

Maintainers should measure entropy.

Maintainers should establish criteria for reengineering as an alternative to fixing.

Evolutionary concerns should be built into the life cycle.

8.5. *Candidate 5*

8.5.1. First statement

Make quality No. 1 (similar to a principle in Ref. [6]).

8.5.2. Discussion

Apart from the obvious characteristic of being a slogan, the statement begs the question of "What are Nos. 2, 3, etc.?" Even if we knew the answer, the candidate would still have the effect of hiding a trade-off. Furthermore, in any particular case, the definition of quality is context-dependent and achievement may be more or less difficult to judge. Finally, the statement is not true enough to be fundamental; after all, one can imagine situations where factors other than quality, profitability for example, might be preferable. All considered, the candidate statement might be better regarded as a *moral, ethic* or *value*. Nevertheless, a restatement of the candidate is possible.

8.5.3. Restatement

Articulate the desired characteristics of quality and explicitly trade among them.

8.6. *Candidate 6*

8.6.1. First statement

Everyone must know the big picture and their relationship to it.

8.6.2. *Discussion*

This candidate may even fail the test of being conventional wisdom. It seems to contradict the practice of breaking a problem into constituent pieces and probably represents a style of management rather than a technical approach. Even if effective in some instances, it is probably a short-term remedy to compensate for poor methods of decomposition; in other words, even if effective today, it is probably not enduring.

8.7. *Candidate 7*

8.7.1. First statement

Design without documentation is *not* design [6].

8.7.2. *Discussion*

As originally stated, the statement is descriptive rather than prescriptive. To solve this problem, the candidate can be restated as something like, 'All components and processes should be documented'. The restatement is not without problems, though. It is clearly *not* abstracted from common practice and seems to be based upon the specious presumption that any documentation is worth the cost of its production, More basically, it is not clear that the candidate corresponds to any experimentally validatable underlying concept. A more reasonable statement was proposed.

8.7.3. *Restatement*

Plan for evolution. Document whatever is necessary for the planned evolution.

8.8. *Candidate 8*

8.8.1. First statement

Put technique before tools (similar to a principle in Ref. [6]).

8.8.2. *Discussion*

Once again, we find that the proposed principle hides a trade-off-this time between reliance on tooling and reliance on technique. In discussion, we discovered that different members of the group placed very different interpretations on the statement. Some thought that it meant (1) 'Use techniques rather than tools,' while others thought that it meant (2) 'Select techniques first and then choose tools that fit.' Some claimed that the candidate had some demonstrated validity, for example in techniques such as 'clean room'.

The statement seems to have been formulated in response to failure, but it also seems possible that the failures might have been the result of using inappropriate tools. So, the enduring nature of the candidate is questionable; it may simply be a reaction to possibly short-term inadequacies in tooling. In short, the statement seems more like a 'rule of thumb' than a principle. An alternative was proposed, although its intent seems substantively different.

8.8.3. *Restatement*

Select a combination of tools and techniques suitable to the job.

9. Recommendations of the workshop

As a resuit of the workshop, five observations and corresponding recommendations were formulated:

(1) There is no commonly accepted criteria for the recognition of fundamental principles. Therefore, the criteria developed by the workshop should be published.

(2) There is no consensus on a set of fundamental principles. One of the workshop participants, Robert Dupuis, agreed to organize and conduct a Delphi experiment among a group of software engineering experts using the criteria developed at the workshop.

(3) The formulation of a well-articulated set of fundamental principles is probably a long and difficult process. A follow-on workshop should be organized at the 1997 International Software Engineering Standards Symposium.

(4) Current collections of software engineering standards lack cohesion because they treat the (currently implicit) principles unevenly. Following a suitable articulation of principles, the standards community should ensure that standardized practices can be traced to fundamental principles and that those principles, in turn, can be traced to underlying scientific concepts.

(5) Existing standards may contain some poor advice in the form of suboptimal practices. The standards community should require some form of empirical validation as an integral part of the standards-development process.

References

[1] I. Asimov. The Search for the Elements 1962.

[2] 1996 Forum on Software Engineering Standards Issues. Proceedings can be found at URL: http://saturne.info.uqam.ca/labo_recherche/lrgl/ses96.htm.

[3] W. Humphrey, Software engineering, in: A. Ralston, (Ed.), Encyclopedia of Computer Science, 1993.

[4] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. Corrected Edition, IEEE Standards Press, New York, 1991.

[5] B. Boehm, Seven basic principles of software engineering. J. Syst. Software 3 (1) (1983)

[6] A.M. Davis, 201 Principles of Software Development, McGraw-Hill, 1995.

[7] M. Lehman, On understanding laws. evolution and conservation in the large-program life cycle, J. Syst Software 1(3) (1980).

[8] W. Royce, Managing the development of large software systems, Ninth international Conference on Software Engineering, IEEE Computer Society Press, 1987 (reprint from WESCON '70. 1970).

[9] A. Abran, Teaching software engineering using ISO standards, Standard View 4 (1996) 139–145.