Int J STTT (1998) 2: 1–5

The opinion corner

Paul Feyerabend and software technology

Gregor Snelting

TU Braunschweig, Abteilung Softwaretechnologie, Bültenweg 88, D-38106 Braunschweig, Germany;
E-mail: snelting@ips.cs.tu-bs.de

Abstract. The following contribution is a plea for more stringent methodological standards in software technology. Certain basic scientific principles are often neglected, principles such as the fact that predictions need to be falsifiable. This appears to confirm the theses of the so-called "constructivists" that "objective truths" are in reality just social constructs. It is thus argued here that one needs a stronger empirical foundation for software technology.

Key words: Software technology – Methodology – Theory of science – Constructivism

1 Introduction: so-called constructivism

The official affirmative action report of the Ministry for Science and Arts in the state of Lower Saxony (West Germany) contains a chapter on every field of science. The chapter on physics, mathematics, and computer science – entitled *Shaking the objectivity postulate* – states: "Many so-called scientific facts, which were considered objective, will perhaps be recognized as hierarchy-centered, control-obsessed, life-antagonistic, since subjective-androcentric interpretations" [5, p. 117]. In other words: the so-called laws of nature are male inventions in order to suppress women. Indeed, Newton's recently discovered secret testament reveals that he invented gravity just to prevent his wife from breaking the glass ceiling.

But let's stay serious. The above statements represent just the feminist version of an epistemological view known as *constructivism*, which was co-founded by Paul Feyerabend [2]. Constructivists do not believe in objective knowledge, let alone laws of nature and their (perhaps approximate) cognizability by humans. Instead, they consider the "laws of nature", roughly speaking, to be the results of an agreement, the product of complex communication processes, that is: a social construct. Of course, "construction" of the "natural laws" is driven by interests: scientists who call themselves truth seekers are in fact just protecting their funding.

Feyerabend wanted to demonstrate "how easy it is to tease people in the name of reason"; he denied philosophical foundations of reliable knowledge with his slogan "anything goes". His ultimate verdict could be found in the 1991 *Who's Who*: "All scientists are criminals". Indeed, many educated people in Germany believe that (1) Einstein has proven that everything is relative, and (2) Gödel has proven that nothing can be proved.

Now one could argue that science *works*: airplanes fly, light bulbs glow, atomic bombs explode. How then can the underlying theories be just social constructs? In a recent interview that the science journalist J. Horgan conducted with Feyerabend [3], it becomes clear that Feyerabend in fact did not deny the epistemological power of the natural sciences – he was afraid of the "tyranny of objectivism", which in his perception leaves no space for human creativity and diversity. Only this angst makes Feyerabend's dadaistic epistemology understandable.

2 Construct and reality

In this article, I would like to point out that in *software technology* sometimes things happen in quite the way Feyerabend and the Lower-Saxonian ladies describe them. I will illustrate this statement with some examples, and then proceed to present some methodological conse-

This article appeared in the Journal of the German Computer Science Society [10], and some observations apply solely to the situation in Germany. English translation and adaptation by the author.

quences. But first of all, we have to take a look at some epistemology and its relationship to computer science.

Let us first study the problem of how to distinguish between constructs and reliable knowledge. Some people believe that the essence of science is to *explain* things: it is the task of science to deliver understandable, or even mechanical models of specific circumstances. Indeed, successful theories and methods also generate explanations for the observed phenomena. But explanations alone can never distinguish between constructs and reliable knowledge and hence have little value. Mankind in history has delivered quite a bunch of "explanations": why does the sun rise in the east? Because it rotates around the earth. Why do I feel sick? Because a spirit possessed my body. What will help? Spiritual homeopathy.

The critical criterion is a different one, namely *predictive power*. If my theory predicts that the moon will appear at point x in exactly one year, and one year later this prediction indeed becomes true, then I can trust my theoretical model: it is reliable enough to serve as a foundation for technical systems. In other fields like medicine or psychology predictive power is the critical indicator as well, predictions being e.g., "the medicine will help" or "the rapist will not do it again". It is not a shortcoming that many predictions are only of a statistical nature, such as in quantum physics or psychology: if I can predict that 10 out of 100 persons will behave in such-and-such a way, or that one out of 1000 electrons will tunnel, I have a reliable theory – even though the predictions only apply to collectives and not to individuals.

In order to be able to test predictions, they must be refutable by experiments. This is Popper's well-known *falsifiability* criterion. Predictions which are not falsifiable are, according to Popper, not scientific statements. However, this should not be taken too dogmatically. Popper himself conceded in an interview shortly before his death that falsifiability is not an absolute principle, but a methodological guideline [3].

Often it is argued that human knowledge is limited by (a) the structure of the cognitive apparatus itself and (b) social context (e.g., education, prejudices, imprinting, conditioning). Indeed, context-dependent statements cannot fulfill the falsifiability criterion; context dependency is just typical for constructs. But those who deny the possibility of context-independent human knowledge do in fact advocate airplanes which are carried by their builder's belief systems, and not by the laws of aerodynamics. This we have seen before – in the Middle Ages.

The first argument however carries more weight and was first put forward by Kant. Today we know that indeed human perception is actively constructed by the brain. But this does not imply that there is only loose coupling between the "inside" and "outside" world. *Evolutionary Epistemology* [9, 12], as introduced by Lorenz and Popper, correctly points out that the human brain has developed in adaptation to reality – if our cognitive apparatus made too many false predictions, mankind would not have survived. Hence it is certainly true that, say, a bat experiences a room completely differently from a human. But it is also true that bat and human develop just different mental representations of the same real spatial topology.

3 Computer science and epistemology

3.1 Abstraction as a standard tool

Now what about computer science? Computer science is not a natural science, because it partially creates its own reality itself. Computer scientists invent or construct abstract concepts or devices, which in turn enable (or limit) the specific applications. In Germany, it is popular to call computer science and mathematics "structural sciences"; recently, however, computer science is seen more as an engineering science.

The ability to find good abstractions is in fact an outstanding skill of computer scientists. Every algorithm already is an abstraction (of specific input values). In software technology, one might think of abstract data types, software architectures, object-oriented design, generic OO-classes, design patterns. Abstractions must not only be sound, but should also be elegant. Experience from the natural sciences indicates that elegant theories have a higher correctness probability. This is the principle of Occam's razor: "Do not introduce superfluous concepts" (counterexample: "an operating system is a 17-tuple").

3.2 Prediction and experiment

What is the computer science version of predictions and falsifying experiments? Predictions in computer science are (as in other fields) obtained from abstractions and theories. Every computer scientist knows statements about the behavior and complexity of algorithms, which are obtained by analytical methods (program verification, complexity analysis) from the algorithm text. Such statements are predictions about the behaviour of an algorithm, when implemented as a program. They are however not always experimentally validated, and some computer scientists in fact consider this needless. For example, Dijkstra only reluctantly concedes that programs may be executed at all; the task of the computer scientist is, according to Dijkstra, to formally verify an algorithm he has constructed [1].

In practice, however, a program runs in a specific technical environment, and therefore a specific form of experiment – *testing* – is indispensable. For formal correctness describes only part of the algorithm's behaviour, just like Newton's mechanics only approximately describes celestial systems. Often, theoretical assumptions, such as correct compilers or Euclidian spatial geometry, are not really valid. Even complexity statements such as "$O(n^{2.81})$ is better than $O(n^3)$" do not hold in practice if the break-even point is beyond tractable problem sizes. Tests, in particular field tests, can uncover errors in specification, design, verification, and interaction with the technical/sociological environment. (The popular method of using tests to find implementation bugs can however in principle be replaced by correctness proofs.)

But testing, like all experiments, can only be used to refute a specific prediction: Dijkstra's dictum "Testing can only demonstrate the presence of errors, not their absence" is a special case of Popper's falsifiability principle.

Recently, predictions are generated by more sophisticated tools. Model checking is a good example: model checking is a method to check a (finite-state) machine against a (formal) specification, and generates counterexamples if the specification is violated. Thus model checking makes predictions about the machine's behaviour in the real world, and has become a very succesful method for hardware verification.

Other predictions in computer science do not consider specific algorithms, but the development methodology itself. Examples in software technology include:

1. "GOTOs increase code entropy" (Dijkstra)
2. "Strong typing reduces run-time errors" (Wirth)
3. "Good modularization reduces maintenance costs" (Parnas)

The advantages of GOTO-free, strongly typed, modular or object-oriented software development are not immediately visible – on the contrary: usually costs increase in the early stages of adoption. Hence the above statements have been heavily attacked when published. But only few explicit falsification attempts have been made such as Knuth's defense of the GOTO or Brooks' defense of process against modules. Both attempts failed (see e.g., Brooks' invited talk at ICSE 95: "Parnas was right, and I was wrong"), and eventually the "weight of experience" – that is, accumulated missing falsifications – led to a general acceptance years later. (Funnily enough, German sociologists have blamed the GOTO damnation to be a capitalist trick in order to squeeze more productivity out of programmers. Perhaps the Lower-Saxonian ladies could explain to them that the GOTO is just an androcentric construct to please control(flow)-obsessed machos.)

3.3 Axiomatic versus empirical approaches

New abstractions in computer science are often introduced in an axiomatic style, but must later demonstrate their usefulness empirically. An outstanding example of this dichotomy is polymorphism in functional languages. Originally created in response to two obviously contradictory requirements, namely type safety and reusability, the Damas–Milner type system guarantees "well-typed programs can't go wrong" even for code which can be used in different application contexts. The axioms are constructed in a subtle way such that typeability remains decidable, even though functions may be used with arguments of different type. While in theory polymorphic type inference is NP-hard, in practice it is efficient. Polymorphism is an important concept today, and Milner received the Turing award.

In cotrast to that, a purely axiomatic approach in mathematics, as exemplified by Bourbaki, hides the fact that mathematics has a connection to reality and is not just a super-smart "Glasperlenspiel" (glass bead game). Neo-platonists such as Penrose even consider entities like the Schrödinger equation to be Kant's thing-in-itself: the unbelievable predictive power of mathematical models cannot, according to Penrose, just be the result of formal sand-table exercises [8].

4 Some observations

4.1 Empirical studies are rare

In 1994, W. Tichy investigated 400 papers which appeared in 1993 in software-technological conferences and journals [11]. He found that

- 43% of all papers did not contain any experiments;
- only 31% of all papers devoted more than 20% of their content to case studies or experimental validations;
- in the IEEE Transactions on Software Engineering 55% of the articles presented new concepts without any empirical validations;
- in the ACM Transactions on Programming Languages and Systems 45% fell into this category;
- only 20% of software engineering papers contained more than 20% of experimental research;
- a comparison with the fields of optical engineering and neural computation showed that 70% of their articles contained a substantial experimental part.

Tichy draws alarming conclusions: "The results suggest that large parts of computer science may not meet standards long established in the natural and engineering sciences". Since computer science is more than half a century old, juvenile immaturity cannot be the reason.

Certainly, big case studies or extensive experiments as in pharmacology can not be demanded from every software technology paper. Even Parnas, Wirth and Dijkstra did not present any experimental data supporting their predictions. Ordinary mortals, sitting in a university computer science laboratory, usually have neither time nor money for experimental validations. And some falsification attempts are just needless: the superiority of graphical user interfaces, compared to ASCII command lines, usually leaps to the eye. But still, there is clearly an empirical deficit in software technology.

4.2 The increasing chasm between theory and practice

The recent article "Strategic directions in software quality" states: "The chasm between research and practice seems particularly wide and increasingly inculturated, to the detriment of both communities. Practice is not as effective as it must be, and research suffers from the validation of good ideas and redirection that inevitably results from serious use" [7].

Industry people like to blame academics that they are spaced out and ignore real problems, while professors like to state that managers cannot tell a paper model from an initial model. Different reasons are put forward to explain this phenomenon. SIGSOFT chair David Notkin believes that often academic researchers sneer at applications, while at the same time industry demands that research should deal with their specific business problems [6]. Tichy refers to missing empirical foundations, which make it impossible for practitioners to evaluate new results.

Indeed, modern software technology is only used in innovative institutions, and not every big software company sticks to well-established software engineering principles. Commercial data processing has largely ignored computer science; instead Cobol and six-week-wonder programmers were employed. Only recently have the (German) commercial mainframers understood that they have a problem, and are considering object-oriented methods, for example.

It is also true that many theoretically oriented researchers have limited interest in applications. But it is not impossible that theoretical results are useful in practice, and the author would like to warn against dismantling fundamental research, just because category theory cannot be transformed directly into dollars.

4.3 Constructivism in software technology

Theory-oriented scientists are wrongfully blamed the sometimes missing the practical relevance of software technology research. Theoretical models (e.g., formal languages) have enormously contributed to the progress of computer science. Neither formal specifications, functional languages, model checking, nor other "esoterica" are the cause of the unsatisfactory state of software engineering. Theoretical models can be checked against reality; many theoretical methods are efficient in practice.

The problem comes in fact from *practical* scientists, who ignore theory, and at the same time avoid empirical validation. Such researchers practice constructivism. An example: software reengineering is a hot topic in software technology. Many reengineering methods employ heuristics in order to reconstruct structural information from old source code. Heuristics always contain free parameters, which must be calibrated using real legacy code. This requires empirical studies. Thus the methodological minimum for papers presenting new reengineering methods is to conduct several case studies. Tool-building actionism alone generates methodological doubts [4].

Anyway, if new methods are difficult to formalize, the least one can expect is that a prototype implementation exists, which can be used for empirical studies. But sometimes software engineering papers present new methods which can neither be described mathematically, nor does an implementation exist.

Correspondingly, not always does the best idea succeed in practice, but the one which has the most market power. In computer science, this phenomenon is stronger than in other fields, partially because – as explained above – computer science is not a natural science; hence there is often more than one technical solution.

But this phenomenon is also due to methodological shortcomings. While Popper demanded that one should try to falsify one's favorite theory every morning during shaving, some software researchers desperately try to "sell" their latest toy, but avoid validation. This leads to a loss in credibility and to constructivist criticism.

5 Some consequences

The preceding discussion argued that the fundamental science-theoretic concept of falsifiability is as valid in software technology as it is in the natural sciences, but is often ignored. In this last section, I would like to propose some practical consequences, in order to improve the situation.

5.1 Theoretical versus empirical research

Contributions which just present an idea, without supplying a mathematical model or an implementation, are of questionable value. Thus methods and tools should be implemented, in order to allow evaluation. (However, university labs are not software companies, hence one should not require professional quality from a research prototype.) Research results must be reproducible, therefore even prototypes should be publicly available. (The *Electronic Tool Integration* platform associated with this journal provides a valuable service to the community in this respect.)

Contributions which introduce new concepts and present theoretical studies are definitely legitimate, if the theoretical model generates falsifiable predictions. Theoretical studies are even worthwhile, if certain phenomena can be more simply described or better explained, even without generating new predictions (that is, possible practical consequences). Theory, however, is questionable in software technology, if it becomes self-contemplation unrelated to applications.

We do not have enough studies about the real value of new theories, methods, and tools. Such studies of course must respect the fact that a new method cannot have the same maturity as an established one. Extensive experiments will rarely be possible, but there will always be enough money for a case study.

Empirical studies can produce surprising results. For example, empirical studies have clearly demonstrated that inspections are an efficient technique for finding flaws in specification, design, and implementation.

Cooperation with industry often makes sense in order to try new methods in a realistic environment. American companies such as IBM or (more recently) Microsoft maintain exchange with fundamental research at universities, to the profit of both sides.

5.2 Scientific activity

It happens that software researchers do not apply their own principles when developing tools. Self-application is an important technique in computer science, and should also be practised in software technology. In teaching, students should heavily exercise modern software technology in student projects. Graduates who practice the art of hacking or have limited programming abilities will not increase our reputation.

Often the same idea is published several times with slight variations (so-called "cut-copy-paste papers"). Correspondingly, there is a proliferation of conferences and workshops, while the quality of many events decreases steadily. Sometimes people who cannot get their papers accepted at major conferences just set up their own conference. Furthermore, citation networks can be observed: papers of competing groups are never cited, while papers of good friends (and of course, one's own earlier contributions) always appear in the list of references.

This phenomenon has its roots in the publish-or-perish setup of today's research business. I strongly believe that the publish-or-perish principle damages science. I have seen tenuring committees which used an "importance factor" for every conference or journal, and then evaluated candidates by counting the publications, each multiplied with the corresponding factor. Such silly procedures would have refused tenure to Newton and Galileo. Quality in research can (and should) be measured, but not by simplistic numbers such as importance factors and citation counts.

Funding policy is another issue. Some funding agencies, such as the German *Deutsche Forschungsgemeinschaft*, have excellent review processes which make reasonably sure that only high-quality proposals get funded. But other (German/European) agencies are easily influenced by political factors, and funding decisions are subject to lobbying. This has a destructive effect on the overall quality of research.

But perhaps the most important factor is the individual researchers themselves. Some scientists are indeed driven by a desire for career and reputation, and not by the desire to improve knowledge and share it with one's students. Career and reputation (which are not a bad thing) will come automatically if one values good scientific principles – political trickery does not necessarily have the same effect.

6 Conclusion

The natural sciences have always been devoted to high methodological standards, and after 2000 years of epistemology we have good criteria in order to distinguish between constructs and reliable knowledge. It would be nice if software-technological results would adhere to the same solidity.

Then, after all, there would be just one remaining task for constructivism: *self-application* (a standard technique in computer science). We leave it as an exercise to the reader to derive the result of such self-application.

References

1. Dijkstra, E.D.: On the cruelty of really teaching computer science. Commun. ACM 32(12), 1398–1404, 1989
2. Feyerabend, P.: Against Method: Outline of an Anarchistic Theory of Knowledge. Atlantic Highlands, 1974
3. Horgan, J.: The End of Science. Reading, MA: Addison Wesley, 1996
4. Müller, H., Reps, T., Snelting, G.: Program comprehension and software reengineering. Dagstuhl Seminar Report No. 204. Also in: SIGSOFT Notes, September 1998, pp. 36–43
5. Frauenförderung ist Hochschulreform – Frauenförderung ist Wissenschaftskritik (*Affirmative action is university reform – affirmative action is science critique*). Ministry for Science and Arts, Lower-Saxony, 1993 (in German)
6. Notkin, D.: Adopting software engineering research: a consumer problem or a producer problem? In: Tracz, W. (ed.): ICSE-19 Window to the World. `http://www.webjammers.com/wow`
7. Osterweil, L., et al.: Strategic directions in software quality. In: Strategic directions in computing research. ACM Computing Surveys 28(4), 1996
8. Penrose, R.: The Emperor's New Mind. Oxford University Press, 1989
9. Radnitzky, G., Bartley, W. (eds.): Evolutionary Epistemology, Rationality, and the Sociology of Knowledge. Open Court, La Salle, 1987
10. Snelting, G.: Paul Feyerabend und die Softwaretechnologie. Informatik-Spektrum, October 1998, pp. 273–276 (in German)
11. Tichy, W., Prechelt, L.: Experimental evaluation in computer science: a quantitative case study. J. Systems Software 28(1), 9–18, 1995
12. Vollmer, G.: Evolutionäre Erkenntnistheorie (*Evolutionary Epistemology*). Hirzel Wissenschaftliche Verlagsgesellschaft, Stuttgart 1992 (in German)