Falsificationism and Software Engineering

Yasuyuki KAGEYAMA*

While Popper's falsificationism has normally been dealt with in the context of such philosophical problems as the rationality of science, I shall consider it here in a practical context, with the help of a model case from software engineering. I shall attempt to find where Popper's falsificationist thought may be applicable to practical concrete problems. My intention in doing so is to demonstrate the suitability of the falsificationist approach to the practical procedures of software development.

1. Controversies concerning Falsifiability

Just as Popper has always been a strong critic of the positivist and inductivist views of science, so has his falsificationism been criticized by logical positivists and those sympathetic to logical positivism. Popper, however, seems to have anticipated such positivistic criticisms from the outset, since most of them are already answered in his first book, *Logik der Forschung* of 1934. Yet from the 1960's, with the integration of the history of science into arguments in the philosophy of science, Popper's theory has been exposed to criticisms quite different from those of the positivists. Thomas S. Kuhn, one of the proponents of the new philosophy of science based on the historical knowledge of science, has criticized the theory of falsifiability in the following way: science has not in fact developed in the way Popper envisaged. Instead there have been theories, such as the Copernican theory and Bohr's quantum theory, which, though initially suffering from several anomalies, were upheld dogmatically by their proponents and finally became successful. It follows that to give up theories simply because of discrepancies with facts is tantamount to giving up science itself. It is, therefore, sometimes necessary to defend theories from the danger of falsification. In other words, being critical is not the same thing as being scientific.⁽¹⁾

One of Popper's main replies to this kind of objections is as follows: "falsifiability in the sense of demarcation criterion does not mean that a falsification can in practice be carried out, or that, if it is carried out, it will be unproblematic. Falsifiability in the sense of the demarcation criterion signifies nothing more than a

*IBM Japan, Ltd.

⁽¹⁾ T.S. Kuhn, *The Structure of Scientific Revolutions*, Chicago University Press, 2nd ed., 1970, p.146, *Essential Tension*, Chicago University Press, 1977, pp. 272f., P. Feyerabend, 'How to defend Society against Science', in I. Hacking (ed.), *Scientific Revolutions*, Oxford University Press, 1981, p. 160.

logical relation between the theory in question and the class of basic statements, or the class of the events described by them: the potential falsifiers. Falsifiability is thus relative to these two classes: if one of these classes is given, then falsifiability is a matter of pure logic — the logical character of the theory in question.”⁽²⁾ What Popper intends to assert is that since falsifiability has nothing to do with actual falsification, no case of unfalsified scientific theories invalidates the central meaning of falsifiability, viz. its logical meaning.

As to the logical meaning of falsifiability, however, Lakatos’ argument seems to be fatal.⁽³⁾ In order to make a falsification conclusive in a logical sense, it is necessary to validate, together with laws of nature and initial conditions, the *ceteris paribus* clause to the effect that there is no other disturbing factor in the world. To validate this clause means to verify, or to confirm it, but, since the clause is a negation of an existential proposition, it is logically equivalent to a universal proposition. And “any strictly universal proposition is not capable of empirical confirmation, so not capable of verification.”⁽⁴⁾ In short, any logically complete falsifiability falls into the same difficulty as the complete verifiability.

Confronted with the Lakatos’ argument, Popper replies: “it is a matter of risky conjecture to which part of a theory we attribute the responsibility for a refutation.”⁽⁵⁾ Yet in saying this, Popper seems to admit implicitly that the logical meaning of falsifiability has something to do with some non-logical factors, in that he tries to save the logical meaning by means of risky conjectures which are of non-logical characters.

Nevertheless, Lakatos’ argument shows no more than that falsifiability is not simply an algorithm only by means of which anyone can decide unambiguously whether a given theory is falsifiable or not.⁽⁶⁾ When it is understood in this way, it

²⁾ K.R. Popper, *Realism and the Aim of Science*, London: Hutchinson, 1983 (hereinafter cited as RAS), pp. xxi. Cf., also K.R. Popper, *The Logic of Scientific Discovery*, New York: Harper & Row, 1965 (hereinafter cited as LSD), pp. 86f., *Die beiden Grundprobleme der Erkenntnistheorie*, Tübingen: J.C.B. Mohr, 1979 (hereinafter cited as GE), pp. XXIX-XXX, *Conjectures and Refutations*, London: Routledge and Kegan Paul, 4th ed., 1972 (hereinafter cited as CR), pp. 41f. Popper had entertained the idea of separating the meaning of falsifiability into the logical and the procedural long before the criticism by Kuhn, and had attempted to avoid the term ‘falsificationism’ since this blurs the distinction between the two meanings (GE, p. XXX.). But Popper himself has used this word (CR, p. 228). Here the term ‘falsificationism’ is used to mean the idea of seeking counter examples rather than favorable ones.

³⁾ I. Lakatos, *The Methodology of Scientific Research Programmes*, Cambridge University Press, 1978, pp. 16-19.

⁴⁾ GE, p. 302.

⁵⁾ K.R. Popper, ‘Replies to My Critics’, in P. A. Schilpp (ed.), *The Philosophy of Karl Popper*, La Salle: Open Court, 1974, p. 1010, pp. 1186f.

⁶⁾ An isolated pure existential statement which is free from any spacio-temporal restric-

will become clear that Lakatos' argument does not imply the total refutation of the theory of falsifiability. If we can regard the process of falsification as iterated procedures, we can continue the attempt of falsifying the *ceteris paribus* clause when this is under suspicion: for the clause prohibits something to happen, just as a universal natural law. To attempt to falsify a *ceteris paribus* clause, we need initial conditions and other *ceteris paribus* clause of higher level. Thus the procedure of falsification has a recursive structure of searching the deeper cause of falsification.

Now, it is not a matter of logic at which level of the recursion we should stop. Insofar as every test is fallible, it ultimately depends upon the attitudes of a theory's advocates whether the theory is to be regarded as falsified or not. Popper has never thought that a falsification immediately leads to the abandonment of a theory, but he has asserted repeatedly that we should not avoid falsifications, and that a theory can be overturned by falsifications.⁽⁷⁾ On the other hand, he does admit that some dogmatism is necessary, and he recommends not succumbing to criticism too easily, in order to defend theories wherever possible.⁽⁸⁾ Yet when should we be dogmatic and when should we be critical? Popper is not explicit on this point. While he has formulated the rules of falsifiability very clearly, he has not elucidated the rules of dogmatic defense of a theory. Popper remarked that any falsification can in principle be avoided, and proposed a rule by which falsifications may be avoided, but he has never proposed a rule obliging the avoidance of falsifications. Criticisms against Popper seem to be concentrated on this point. Popper has been construed as ambiguous concerning the roles of dogmatism and criticism.⁽⁹⁾

One of the reasons why Popper has been so construed is that there are some discrepancies between Popper and his critics on what is to be defended dogmatically. The quest for truth is for Popper the ultimate aim of science which is to be held with conviction, to the extent that any proposed theory, if falsified, can be overturned for the sake of this aim. For his critics, on the other hand, it is a proposed theory that is to be held dogmatically: if necessary, any purported aim can be modified for the defense of the theory, just as the meaning of understanding was changed during the development of quantum mechanics.

tion is unfalsifiable only by its logical form; but such a statement does not, as Popper says, appear in science. Cf., LSD, pp. 69f. In any ways, Popper admits that falsifiability is not a sharp logical criterion. Cf., LSD, p. 82, K.R. Popper, *Unended Quest*, Glasgow: Fontana/Corins, 1976 (hereinafter cited as UQ), p. 42.

⁽⁷⁾ CR, p. 215, UQ, p. 41, p. 79, K. R. Popper, *The Myth of the Framework*, London: Routledge, 1994 (hereinafter cited as MF), p.105.

⁽⁸⁾ CR, p. 49, p. 312, K.R. Popper, 'Normal Science and Its Danger', in I. Lakatos, A. Musgrave (ed.), *Criticism and the Growth of Knowledge*, Cambridge University Press, 1970, p. 55, UQ, p. 42, p. 51, MF, p. 16, p. 94, K.R. Popper, K. Lorenz, *Die Zukunft ist offen*, München: Piper, 1985, p. 60.

⁽⁹⁾ Cf., A. O'Hear, *Karl Popper*, London: Routledge and Kegan Paul, 1980, pp. 108ff.

2. Falsification from the Viewpoint of Software Engineering

In order to clarify the roles of dogmatism and criticism, I propose to employ a model from software engineering. Let scientific theories which are subject to falsifications be software systems or products; let anomalies which afflict scientific theories be systems problems; and let falsifications be errors or bugs of the systems. The pursuit of reliability is to be posed here as an aim of software engineering.

Then, the fact that no theory is exempt from anomalies would correspond to the fact that no software product is exempt from problems. Just as the fact that "[n]o theory ever agrees with all the facts in its domain, yet it is not always the theory that is to blame",⁽¹⁰⁾ it is very often the case that certain problems in a software product are due not to a bug in the program as was first thought, but to errors in another system such as Operating System, or to errors in operating procedure. Even when problems in a product are due to errors in the product itself, the presence of a few bugs does not necessarily lead to the product being shelved. And there are even products such as MS-Windows, initially defective in many areas and unpopular with users, but never discarded by its developers, which through numerous upgrades and improvements have finally prospered. Just as Popper's opponents say that no theory is overturned unless another new theory is accepted,⁽¹¹⁾ so too is a software product not easily overturned; furthermore, because of the problem of compatibility and de facto standards, some old products often remain in use even when a better product is put on the market. Such a case can be seen in the relation between MS-DOS and Macintosh or OS/2. Since most users have accumulated huge software assets, it is very difficult for them to switch to new products.

Falsificationism, which advocates the abandonment of theories by falsifications, seems, therefore, incapable of grasping these facts of software engineering properly. But the falsificationist approach should not be abandoned so quickly. It is striking how the actual situation of software engineering would be greatly aided by the falsificationist approach.

The prevalent view in software testing has been verificationist, that to test a software program is to prove its correctness. But this verificationist approach actually provides very poor results, since proving the correctness of a program tends to involve selecting test data with low possibilities of proving the deficiencies of the program; and the fact that it is impossible to prove the completeness of any programs lowers the motivation of those who test them. Even a program which has been shown to function correctly may contain some serious errors.⁽¹²⁾ For these

⁽¹⁰⁾ P. Feyerabend, *Against Method*, London: Verso, 1978, p.55.

⁽¹¹⁾ Cf., T.S. Kuhn, *The Structure of Scientific Revolution*, op. cit., pp. 77ff.

⁽¹²⁾ G.J. Myers, *The Art of Software Testing*, New York: John Wiley, 1979, pp. 4-7. The test data with a low probability of showing malfunction of a program are also the

reasons, it is worth adapting a falsificationist approach to the effect that “[t]esting is the process of executing a program with the intent of finding errors.”⁽¹³⁾ That is, the best way to vindicate the conviction that a program is perfect is to “try to refute it, that is, to try to find out imperfections rather than just confirm that the program works correctly for some set of input data.”⁽¹⁴⁾

It may be objected that a prototype system, which is doomed to be finally overturned, does not need to be tested so severely. Our reply to this is that a program should be tested as severely as possible because we do not, and cannot, give up the program itself. The more important the program, the more severe should be the test, so that the falsificationist attitude will be all the more significant. In other words, the degree of severity of the falsificationist attitude toward a software product is proportional to the degree of commitment to it.

From an engineering point of view, however, what is of practical importance is not whether a product is falsifiable (testable) in the sense of the philosophy of science, but rather how effective the process of detecting the cause of errors is. As Lakatos’ argument suggests, it is often pointed out as a critique against falsificationism that it is sometimes impossible to determine unambiguously which part of a theoretical system is responsible for errors; yet in the case of software engineering, the cause of errors cannot be left open. Though the suspect program may turn out to be free of error, it still remains defective until the real cause of error is identified. Hence software engineering is, for the most part, concerned with the efficacy of detecting errors rather than with the decidability of a test.

As the test of a software product is, of course, itself fallible, it is ultimately a matter of choice whether a putative bug is accepted by engineers as a genuine error in the product or not. Even so, we must design our program in advance so as to make it easy to detect errors in case of problems. This is to reduce the risk of the ‘risky conjecture’. Detecting the cause of an error is efficient if a program is so structured that the search space narrows in proportion to the increase in iteration of recursive attempts to falsify hypotheses concerning the cause of the error. Most of the contemporary methodologies of software design are based on this concept.

We are now in a position to clarify the role of dogmatic and critical attitudes by means of the engineering point of view set out above. In order to improve a product, it is necessary to take a thoroughly critical attitude toward it; but the

data with a high probability of showing its correctness; this may correspond to a hypothesis with a high probability of being confirmed, but lacking empirical contents.

¹³⁾ G.J. Myers, *The Art of Software Testing*, op. cit., p. 5.

¹⁴⁾ G.J. Myers, op. cit., p. 7. In the sense that “[t]esting cannot show the absence of defects; it can only show that software defects are present” (R.S. Pressman, *Software Engineering*, New York: McGraw-Hill, 3rd ed., 1992, p. 597.), there is an asymmetry of verification and falsification in software, which is adduced by Popper as an argument for falsifiability.

product itself may be held dogmatically. Or rather, the critical attitude toward a product so necessary for its improvement depends upon a prior dogmatic commitment to the product. There would be no need to be critical about a product that could be easily given up; for it is a product which recognizably lacks the capacity for improvement which is readily abandoned. The testing of a product should be severe if the aim is not to reject the product, but to improve it.

Dogmatic attitudes precede the proposal of theories or the development of software products.⁽¹⁵⁾ It is not only impossible for us to avoid such dogmatism, but it is also very difficult even to be aware of it. Nevertheless, it is precisely because of this dogmatism that we need critical examination to induce awareness of our dogmatism. But critical examination does not entail the abandonment of our theory. We are intent on improving our theory; and to improve a theory is quite different from abandoning it. It should be noted that thoroughgoing fallibilism implies not only the fallibility of a theory, but also the fallibility of tests conducted on the theory; it would be inappropriate to overturn a theory because of the results of tests which are fallible. It follows that one of the consequences of fallibilism is to defend a theory in face of any alleged falsification.

3. The Viewpoint of Piecemeal Engineering

Since this is the case, we may now ask ourselves what is it that prevents Popper from being clear about the roles of dogmatism and criticism.

So long as Popper thinks that in principle it is possible to overturn theories in the natural sciences, his attitude is not that of a participant in a competition, but that of a referee or of a bystander. This may also be seen from his evolutionary epistemology. For Popper, the question whether one's own committed theory can survive is not an issue. His dictum that "we can let our theories die in our place"⁽¹⁶⁾ is surely of high ethical import considering that men in the past were eliminated with their theories; but this way of thinking might also lead him to assume that theories can be discarded time and again. If theories were not so easily separated from those who developed them, Popper would have never said that theories may be eliminated; for then to eliminate theories would mean to eliminate their adherents. Engineers do indeed occasionally abandon their own products; but, as we shall see later, it is not so easy to separate software products from their producers.

From these considerations we can see that there is consciousness of the concerned behind dogmatic attitudes. Though Popper's methodology of the natural

¹⁵⁾ Cf., UQ, p. 51.

¹⁶⁾ K.R. Popper, *In Search of a Better World*, London: Routledge, 1992 (hereinafter cited as SBW), p. 28., MF, p. 7, p. 69.

sciences seems to lack this awareness, it can be found in his philosophy of society and social sciences ; that is, in his account of piecemeal social engineering.

Against Utopian social engineers who advocate selecting the ultimate end at the very beginning and, using this end as their blueprint in a programme of drastic change to our society with a view to constructing the ideal society, Popper offers the following objections. We have insufficient knowledge to perform grand scale social engineering, and we are unable to see the whole of our society at one time. Since no social conduct can produce results exactly as predicted, it is impossible to judge whether a widespread reform is approaching to the chosen end or not. It is by reforms in small steps that we can learn from our mistakes.⁽¹⁷⁾ In deploying these arguments, Popper shows a correct grasp of the spirit of engineering. For the development of a software system, the waterfall development model is still the method usually employed. This method prescribes defining in the first place all the requirements of the system to be developed ; the whole system is then to be designed according to these defined requirements from the top down ; there follows coding, testing, and finally the completion of the system. But this model causes many problems. Software systems are intrinsically highly complicated entities,⁽¹⁸⁾ to the extent that even a renewal of the system may result in subtle deviations from the requirements as they were first envisaged. Because customers scarcely know what they want in the first place, they tend to modify and broaden their requirements during the system's development. The Utopian way of thinking exemplified by the waterfall model is doomed to create hell on earth in its very attempt to establish heaven.⁽¹⁹⁾ The realization has led to the construction of many new models which advocate an incremental development of software systems.

A society, in the same way as a software product, cannot be replaced by another one all that easily. It can still be improved, but only by proceeding one step at a time in a deliberately incremental way ; only piecemeal improvement is possible.

When Popper says that falsified theories may be overturned, he appears to ignore his view of piecemeal social engineering. Insofar as he thinks that a theory can be overturned as a whole, his thought has some similarity with that of Utopian social engineers. Yet although any individual social policy may be overturned when it fails to achieve its aim, the citizens comprising a society are never suscep-

¹⁷⁾ K.R. Popper, *The Open Society and Its Enemies*, London : Routledge and Kegan Paul, 4th ed., 1962 (hereinafter cited as OS), vol. I pp. 157-164., *The Poverty of Historicism*, London : Routledge and Kegan Paul, 1957, p. 67.

¹⁸⁾ F.P. Brooks Jr., *The Mythical Man-Month*, Reading : Addison-Wesley, 1995, pp. 181-186. The fact that 20-50% of bug fixes give rise to new bugs (F.P. Brooks Jr., op. cit., p. 122) manifestly demonstrates the complicatedness and invisibility of a software product.

¹⁹⁾ OS, vol. II p. 237.

tible to elimination. However faulty a society may be, it cannot be demolished as easily as Sodom or Gomorrah. One of the aims of piecemeal social engineering is to avert social catastrophes, and this aim implies, in a sense, a dogmatic commitment to one's society. Hence it can be said that as to the practical matter of social reform, Popper's moral decision: "[i]t is better to suffer injustice than to commit an injustice"⁽²⁰⁾ induces him to take the position of deliberate piecemeal engineering, just as Abraham tried to save Sodom and Gomorrah for a few good men in those cities. But as to the theoretical matter of evaluating scientific theories, his evolutionary point of view according to which any theory is dispensable rather than us prevents him from taking an engineering point of view.

4. Instrumentalism and Engineering

Inasmuch as a software product is a kind of instrument, there arises a question concerning the relation between falsificationism and instrumentalism. One of the reasons why Popper's methodology of natural science is not so prudent as his piecemeal engineering is that he has been attacking instrumentalism fiercely for such a long time; that is, his criticism of instrumentalism is another factor preventing him from taking an engineering point of view.

When he says that "it hardly makes sense to say that we submit an instrument to the severest tests we can design in order to reject it if it does not stand up to them",⁽²¹⁾ Popper seems to deny the significance of testing instruments. It is admissible that an instrument is more conventional than a theory in that requirements for an instrument are sometimes canceled when it fails to pass a test, and that "[i]nstruments, even theories in so far as they are instruments, cannot be refuted."⁽²²⁾ But both theory and instrument are capable of improvement. A test is more important for an instrument than for a theory in the sense that the former can be seen to conflict with reality more easily than the latter.

It is often the case that during the search of causes of a software problem, the problem itself happens to vanish incidentally. This can make it very difficult to achieve a complete explication of the mechanism for the erroneous phenomenon, and the attempt to explicate may be aborted. Ad hoc evasions are occasionally undertaken for the sake of saving time and cost. Such a maneuver, typically based on an instrumentalist idea, is more often than not bound to fail. Repeating ad hoc evasions without recognizing the true cause of errors inevitably leads to hideous complication in a software product, and makes it even more difficult to discover any

²⁰⁾ SBW, p. ix.

²¹⁾ CR, p. 113., cf., RAS, p. 114.

²²⁾ CR, p. 113.

causes of errors ; it results in what is known as spaghetti code. Software engineering is concerned with the search for truth ; for long term reliability cannot be obtained without searching for the true causes of errors. In the sense that an engineer tries to solve problems, he is searching for truth.

It seems that a pragmatic character of falsificationism has not been observed in this way before because of Popper's intensive claim for the ideal of truth and his fierce criticism of instrumentalism. But it is not unnatural to evaluate the pragmatic character of falsificationism when considered from an engineering point of view. The method of searching, not for confirmation, but for falsification is useful both in the search for truth and in the search for reliability.⁽²³⁾

5. Engineering and Puzzle-Solving

One might say that what has been described so far is completely understandable only in terms of Kuhn's puzzle-solving in normal science. This is admissible. I think moreover that Kuhn's model of normal science is more suited to engineering activities than to scientific activities. For instance, although the existence of a paradigm in a normal science does not necessarily mean the presence of explicit rules,⁽²⁴⁾ such rules can be found in manuals of any software development communities as a standard of the methodology for software development. Those rules of standard methodology and the fundamental theories of computer sciences are completely beyond dispute. Just as Kuhn claims, against Popper's methodology, that " it is the individual scientist rather than current theory which is tested " ,⁽²⁵⁾ it is rather the engineer's skills and abilities which are regarded to be responsible for the failure of a development project. And the larger the project, the more the fundamental architecture must be unified and standardized, and the more conceptual integrity is required ; otherwise, the project management would be very difficult.⁽²⁶⁾

These facts conform very well to Kuhn's model ; but the very conformity of actual projects in software development to Kuhn's model is itself a problem. Since the causes of errors in software products, as opposed to those in scientific theories, are easily identified, it is not so difficult to attribute a cause to an individual engineer.

²³⁾ Popper argues that his critical approach is also effective in the search for reliability : he adduces the high security of airplanes which, he says, is said to be due to the examination of accidents ; i.e., it is the result of learning from errors. K.R. Popper, 'The Critical Approach versus the Mystique of Leadership', *Human Systems management* 8, 1989, p. 265.

²⁴⁾ T.S. Kuhn, *The Structure of Scientific Revolution*, op. cit., p. 44.

²⁵⁾ T.S. Kuhn, *Essential Tension*, op. cit., p. 271.

²⁶⁾ F.P. Brooks Jr., *The Mythical Man-Month*, op. cit., pp. 42-50.

For this reason, there remains the notorious effect of errors upon the evaluation of engineers. Furthermore, a software engineer tends to think of his product as an extension of his ego.⁽²⁷⁾ Then a fault in his product will be his own fault. Yet all will be well if no fault is found in his product. Hence the tendency to neglect errors; and eventually the tendency to conceal them.

“A programmer who truly sees his program as an extension of his own ego is not going to be trying to find all the errors in that program. On the contrary, he is going to be trying to prove that the program is correct — even if this means the oversight of errors which are monstrous to another eye.”⁽²⁸⁾ A test to prove the absence of errors can easily be achieved and would accordingly be a poor test. Needless to say, the errors thus overlooked will result in a multitude of disastrous problems. Even though a software product is designed to be amenable to easy error detection, the error detection process may be interrupted by an engineer who happens to find an error that may reflect badly on his own responsibility. And the deadline factor may intensify the tendency to treat errors ad hoc.

These problems are due not only to the psychological factors of software engineers, but also to the structural similarity between software development and bureaucratic puzzle-solving in a normal science. One of the fundamental traits of a bureaucratic activity is that all tasks are done steadily in a mechanical way whereby efficiency is obtained. No error may be allowed to thwart the smooth flow of tasks. Thus any error-causing factor of any kind will be excluded from the outset, and to avoid giving rise to errors, every individual engineer tries to keep the situation as it is.

There are some rules designed to prevent these pernicious tendencies, such as the separation of testers and programmers, or the preparation of the test results before the actual tests take place. But there remains the problem of the individual engineer's attitude. If errors were admitted and eliminated in a constructive way, no problem would arise. In practice, however, few errors are permitted, and engineers who fail are eliminated together with their errors. Such a situation encourages the concealment rather than the detection of errors. As Popper says, “[i]t is like being eager to throw good money after bad.”⁽²⁹⁾

²⁷⁾ G.M. Weinberg, *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold, 1971, p. 54.

²⁸⁾ G.M. Weinberg, op. cit., p. 55., cf., G. J. Myers, *The Art of Software Testing*, op. cit., p. 5. This phenomenon is the same as the fact that a man who has collected every car advertisement prior to purchasing a new car tends to see after his purchase only the advertisement for the car he bought, and is blind to the others; his behavior is explained as a way to avoid being informed by other ads that he is not a wise buyer (cf., G.M. Weinberg, *ibid.*). This attitude is extremely similar to that of Marxists who seek only confirmations for their view when reading a newspaper. Cf., CR, p. 35.

²⁹⁾ UQ, p. 34.

To break the deadlock, a reconstruction of the value systems of engineers would seem necessary. For example, testing a program is usually regarded by ordinary engineers as petty, trivial work ; Popper's assertion that " [e]very refutation should be regarded as a great success"⁽³⁰⁾ may indicate how such attitudes could be changed. In a sense, reliability of a product depends upon its producers being truly critical. The tendency to neglect the errors of a product, whether consciously or unconsciously, must be countered by encouraging a positive attitude toward evaluating the detection of errors.

6. Critical Attitudes in Technological Activities

It might be objected here, however, that even if the critical falsificationist approach is of vital importance in improving the reliability of software products, its value is restricted to normal scientific activities such as software development ; to attempt to apply this critical falsificationist approach to the fundamental framework of the activities as well would harm the steady flow of bureaucratic activities. It is a fact that fast growing bureaucratic organizations have few members who risk adopting a critical attitude. More specifically, the bureaucracy of an organization attaining success in a certain business will grow rapidly in order to maximize the profit from that business. This is because everyone wishes to participate in the success. And a certain measure of bureaucratic framework is indeed necessary, since incidental success unsupported by a bureaucratic organization will quickly vanish.

Nevertheless, to be critical toward the fundamental framework of technological activities is still of vital importance, since the technological environment changes rapidly. A bureaucratic organization has a fundamental propensity to repeat previously successful procedures ; it tends to overlook anomalies which cannot be captured by the procedure. Those few engineers who recognize the anomalies tend to be expelled from their own community, because they are not amenable to puzzle-solving. These expelled engineers may then create new technological trends in another community ; and the original community which expelled them may itself succumb to the new trend. If, therefore, the survival of a particular community is not important, an engineer could dispense with his critical attitude ; he could continue to develop the technology, but in another community. He would only require a critical attitude if the survival of his own community mattered to him.

It may be easy today to criticize an inflexible framework ; for many business communities have proved willing to restructure themselves for innovation after the 1980's when technological changes intensified. Yet even this restructuring is open to criticism : bureaucratic restructuring for innovation in the face of new technologi-

³⁰⁾ CR, p. 243.

cal trends is occasionally liable to be Utopian. Pluralism in software technology, an area full of uncertainty, seems desirable ; yet bureaucratic technological prescriptions attempt to unify their activities into a single pre-established architecture or methodology, in the course of which, most previously existing frameworks disappear. Sweeping innovations of this kind are neither supported by all of the engineers, nor guaranteed to be valid, but to do otherwise is not permitted. Just as Popper says, “[t]he reconstruction of society is a big undertaking which must cause considerable inconvenience to many, and for a considerable span of time. Accordingly, the Utopian engineer will have to be deaf to many complaints ; in fact it will be part of his business to suppress unreasonable objections.”⁽³¹⁾

The tendency of bureaucratic innovation to be uniform may be explained in part as an attempt to catch up with technological changes as quickly and as completely as possible ; yet, in the computer industry where only mottoes tend to be too ahead, the innovations are sometimes superficial and insufficiently checked. Such innovations may vanish as soon as the boom is over. Thus a critical attitude needs to be adopted not only toward rigid frameworks, but also toward Utopian innovations. This may be very difficult advice to follow ; by supporting innovation one can easily acquire a reputation for being progressive, while opposing innovation makes one appear conservative. Since the right direction on a new technology can only be judged by hindsight, to be critical may be a high risk strategy, whether in support of, or in opposition to, an innovation. Few people are willing to take the risk.

Thus most activities in software engineering are actually very similar to those in normal science, and critical attitudes are scarcely adopted or maintained. Nonetheless, “something rare ...may be far more important than something going on all the time.”⁽³²⁾ We might say that a falsificationist critical attitude has rarity value in software engineering.

³¹⁾ OS, vol. I, p. 160.

³²⁾ J. Watkins, ‘Against Normal Science’, in I. Lakatos, A. Musgrave, *Criticism and the Growth of Knowledge*, op. cit., p. 32.