On formal and informal reasoning in program construction

R.J.R. Back, Åbo Akademi

To Jaco

The problems with constructing and maintaining large programs are well known. The low quality of software is certainly one of the main stumbling factors in the computerization process, and one that has caused more frustration than any other aspect of this technology. Although the problem is so well recognized, it seems very difficult to cure. Different proposals have been put forward, such as higher level specification and programming languages, a better organization of the program construction process and quality control measures, improved testing methods and more formal methods for program construction. It is this last proposal that we will study in more detail in this short note.

Before we look at the formal methods, to see whether they promise a solution to the software quality problem, let us turn the question upside down. Rather than ask why the present informal program construction methods do not work well enough, we ask why they do work as well as they do. The programs may be bugridden and difficult to maintain, but for most cases they do work. There must be something in the present informal methods that does go quite a long way towards the goal of quality software. This suggests that the informal methods are good in themselves, but that they are not sufficient for achieving very high levels of quality.

The approach we want to put forward here is that the informal and the formal approaches complement each other, in a way that is in close analogue to the usual way in which everyday science is conducted. Our starting point is the view of scientific work as it is done in e.g. the natural sciences, where empirical and theoretical studies form a close symbiosis. This view can be described roughly (and somewhat naively) as follows.

1. Understanding a new phenomenon in e.g. physics starts by empirical observations. The phenomenon is identified and its manifestations are studied by empirical experiments. These experiments are carried out until some kind of general picture emerges, which provides some (often partial) explanation of what is going on. The experiments are done against a body of prior knowledge, both theoretical and experimental, which guides the experiments. In other words, the experiments are not done in a vacum, but experimenters have a reasonably good idea of what they are looking for.

2. Based on the initial experiments, an explanation of the phenomenon is attempted. This is a generalization, i.e. a model that abstracts away from the specific cases that have been studied in the experiments to provide a more general description of the phenomenon. The model also predicts the outcome of experiments which have not yet been made.

3. This general explanation is then tested against the reality by performing new experiments. The outcome of these experiments is predicted by the generalization. The new experiments should be chosen such that the possibility of falsifying the model is as big as possible. In this way erroneous generalizations can be dispensed with quickly.

4. The new experiments often do not totally falsify the generalization, but point at weaknesses in it. Based on these new experiments, the general explanation is modified so that the new experiments are also taken into account. The modified explanation is then again tested experimentally. The cycle is repeated until there is no contradicting evidence from experiments. The testing of the hypothesis does not have to be made by the same researchers, it may be done by different research groups, and the activity may be stretched over space and time (even over very long periods, like years, decades and centuries).

Different philosophical theories of science differ in how this enquiry is carried out: when a hypothesis or theory can be considered confirmed, how the experiment should be tested and so on. When certain assumptions about the physical reality are satisfied, these questions can e.g. be analyzed by statistical means. The theories agree, however, on the need for the duality between theory and experiments, and on the different nature of these two activities. Theory is *deductive* while experimentation is *inductive*. A theory should be internally consistent, and it should be built according to standard mathematical requirements of rigour and exactness: the definitions should be exact and clear, the theorems should be well defined and the proofs should be rigourous. The theories must have an interpretation in the real world to be useful, but they must also by themselves form a consistent and rigorously developed mathematical theory.

Experimentation studies the phenomenon as it manifests itself in reality. It is often carried out against a body of theory, so it is usually aimed at testing some hypothesis by which the theory is extended, but not all of it. Experimentation seems to have two different purposes: on one hand it is used as an inspiration for formulating new theories and explanations (hypothesis), on the other hand it is used to test the validity of existing hypothesis, formed on the basis of previous experiments or derived (deductively) from hypothesis formed on the basis of previous tests.

Informal methods in program construction Let us now consider how the above relates to program construction. Our first observation is that informal program construction, as it is done in practice, works because it applies the experimental-deductive scientific method. In constructing a program, one thinks in terms of specific examples (test cases). These examples are generalized to a program. The program is a true generalization, because it is intended to also apply to other cases than those considered explicitly. After constructing the program, it is tested on a collection of test cases. These form the new experiments. Usuallly the test cases are constructed so that the likelyhood of detecting an error is as big as possible, i.e. the purpose is to refute the hypothesis that the program works correctly. If a test reveals an error, the source of the error is located (a bug), the hypothesis is revised (the bug is fixed) and testing of the new hypothesis is started. This process is repeated until no further errors are detected, in which case the program is accepted as correct (delivered to the customer), i.e. the hypothesis (or theory) is considered to be confirmed.

Program construction is thus a classical example of the experimental scientific method: the interplay between experiments and theory. Even the criteria for accepting a hypothesis are the same, the hypothesis is accepted when all experiments confirm it. Also, this acceptance is conditional, as it is recognized that further experimentation may still invalidate the hypothesis (as it usually does). The informal program construction method does indeed work in practice, for the same reason that the experimental scientific method works in general: there is enough regularity in the phenomen studied that a reasonably careful testing gives a good basis for assuming that the theory is valid for most cases that can occur in practice.

Formal methods in program construction After having convinced ourselves that the practical approach to program construction rests on good, solid scientific methods, let us now turn to the issue of formal reasoning. The main question here is whether we can do better than this. Our first observation is that the analogy between computer programs and the physical reality that natural scientists study is not complete: the physical reality is given and we only know about it through our experiments, whereas the program is constructed by ourselves. In other words, we have complete information about the program and its working, whereas we do not have complete information about the physical reality. Taking the experimental scientists approach to program construction therefore ignores information that is available. The program that we ourselves have constructed is treated as a black box, the inner workings of which is unknown to us. Or it is treated as a white (or transparent) box, whose inner workings can be seen, but where the function is so complex that it cannot be understood sufficiently to be able to predict the outcome in general.

The analogy we are looking for is maybe then not the way in which the physicist construct theories, but more the way in which mathematicians construct theories, because the latter are in the same situation as we are: they are working in a reality that they have constructed themselves. The problems in mathematics come from a similar source as in programming: the reality that has been constructed is so complex that its working cannot be understood in full detail at once, even if it is in fact known in complete detail.

The way in which mathematicians prove theorems might be a good starting point. We can consider a program as a theorem: it asserts that a certain algorithmically defined function satisfies certain desirable properties. In mathematics we have an informal and a formal way of reasoning. The informal way consists in constructing special cases or examples which illustrate the phenomenon being considered. From these examples a generalization is then made, which again covers more cases and examples than what has actually been considered. These generalizations (lemmas, definitions etc.) are then used in the proof. The proof itself proceeds by formal reasoning. If the theorem can be shown to be correct by formal (or mathematically rigorous) reasoning, then it is accepted as correct. If, however, the proof cannot be constructed, then the usual recourse is to analyze the difficulty by trying to construct new examples and cases where the difficulty is identified. Based on this experimental study, the generalizations are modified so that the difficulties are avoided. This process may be continued until the proof succeeds or until a counterexample is found which finally kills all hopes that the theorem could be fixed to be true.

Comparing this approach to the purely experimental approach by the physicists, we see one major difference. The confirmation of the hypothesis is not done by testing but by deductive reasoning. In other words, the hypothesis is accepted if it can be proved rigorously that it is correct. Only if the proof does not succeed is the experimental stage re-entered. Hence, the role of experiments and testing in mathematics is not to confirm a hypothesis but to generate new hypothesis or to refute the given hypothesis by counterexamples. Still, the experimental part of the mathematical work is very important, and often provides the inspiration and the real workbench where mathematical discoveries are made.

A view of program construction How would we apply this same methodology in the construction of computer programs. First of all, we need to recognize the important role that experimentation plays in program construction, also in the case where the goal is to construct programs that are formally verified correct. In the same way as in mathematics, it provides the playground on which inventions are made. These inventions are based on intuition, i.e. based on the understanding of the problem that has been gained from analyzing a collection of examples, in combination whith the general understanding that has been gained previously. In part it can also arize from purely formal manipulations, based on the mathematical structure of the hypothesis and the formal deductions that can be made from it. However, the first source of inventions would seem to be the more important one, at least as long as the mathematical theory of program construction is as rudimentary as it is today, with few powerful theorems and results to rely upon and no uniform basic theory agreed upon.

The second issue is to recognize the role of verification in program construction: It is the only tool available for confirming that the program works correctly. Difficult as it may be, we cannot hope for achieving greater confidence in the correctness of our programs unless we can verify mathematically that they are correct and have some kind of machine checking of the proofs.

Program construction thus takes place in two parallel but separate worlds, the *experimental testing world* and the *deductive confirmation world*. The interplay between these two worlds occurs in most program construction situations, from direct verification that a program meets its specification to program construction by stepwise refinement and the construction of precise specifications for some required sofware product. To only have experimentation and restrict oneself to experimental confirmation of program correctness is to settle for less than is achievable. To only accept formal reasoning is to deny the main source of inspiration and invention that underlies the construction of efficient and useful programs.

The restriction to experimentation alone is common and even prevailing today, and is usually based on the view that program verification does not work in practice anyway, at least not for larger programs (an issue that cannot be considered resolved yet). The restriction to formal methods only has not to my knowledge really been put forward seriously.A somewhat less extreme but still one sided position is, however, to accept that the informal testing/experimentation activity is needed, but that it need not be recorded in the development process: all experimentation should be done behind the scenes and only the formal reasoning should be recorded. This position, analogues to Gauss' view on mathematical proofs, leads to terse and unintuitive presentations of program constructions. (Often this is only a question of how the material is presented and not an explicitly taken standpoint).

Not recording the intuition and experimentation behind the construction makes the the programs, proofs and program derivations difficult to follow and understand. In this one follows established mathematical tradition: the definitions and proofs are often presented without explaining the intuition behind them. This makes it difficult for people who try to get into a subject to understand what is being done and why. The analogy with poorly documented program text should be obvious here. The need for giving the intuition is felt most strongly in textbooks in mathematics and, as a consequence, the purely mathematical treatment is extended with intuitive explanations, examples and counterexamples as well as exercises. All these serve the same purpose, to present the experimental background that has lead to the formulation of the theory.

A consequence of not recognizing the important role of experimentation is that the need for methods and tools which support the experimentation phase of formal program construction is also not recognized. Consequently no tools are built for this. Such tools can be constructed today; the modern workstation environment with its powerful graphic tools should provide a good workbench for experimentation. The important aspects of such an environment is that it should be very flexible and strongly visually oriented. It should encourage both informal experimentation and formal deduction and verification. It should also record both these activities in a consistent and structured way, so that the necessary documentation is created automatically during program construction, as a byproduct of it.