On B

Jean-Raymond Abrial

Consultant,
26, rue des Plantes, 75014, Paris.
abrial@steria.fr

In the B-Book [Abr96], an introduction entitled "What is **B** ?" presents it in a few pages. There is no point in reproducing this introduction here. There is, however, clearly a need to have in this book a sort of informal presentation of **B** complementing that of the B-Book for those readers who are not familiar with this approach. This is the purpose of this short text, where the emphasis will be put on the question of the development process with **B**. At the end of the text, I will also cover some more general problems concerning **B** (tool, education, research, future).

B *is a "Debugger".*

The aim of **B** is very practical. Its ambition is to provide to industrial practitioners a series of techniques for helping them in the construction of software systems. As everyone knows, the complexity of such systems, as well as the context within which they are developed makes them very prone to errors. Within this framework, the goal of **B** is to accompany the system development process in such a way that the inevitable errors, produced either during the technical specification phase, or the design phase, or, of course, during the coding phase, that all these errors are trapped *as soon as they are produced*. In a sense, from that point of view, **B** is nothing else but a generalized debugging technology.

Classical Debugging Techniques are Based on Execution.

In classical software developments, errors are usually (but partially) discovered, and hopefully corrected, after the coding phase: this is done by checking the final product (or better, some parts of it) against a variety of *tests* supposed to cover the widest range of behaviors. Another technique, which becomes very popular these days, is that of *model checking* by which it is shown that the final system satisfies certain properties: this is done by an exhaustive search under all possible executions of the program.

As can be seen, both techniques, testing and model checking, work on the final product and are based on some "laboratory" *execution* of it. Since we believe in the great importance of trapping errors as soon as they are produced, it is clear that such techniques cannot be used during the technical specification and the design phases, where no execution can take place.

The **B** *Debugging Technique is Based on Proofs. Conceptual Difficulties.*

The debugging technology, which is proposed by **B**, is thus not based on execution, it is rather based on *mathematical proofs*. This apparently simple approach, which is widely used in other engineering disciplines, poses a number of specific problems.

When execution is the criterion, the developer obviously tends to write his formal text (his program) with *execution in mind*. He reasons in terms of data that are modified by some actions (assignments), and he constructs his program by means of a number of operations on such actions: conditionals, sequencing, loop, procedure calls, etc.

With **B**, the developer (at least, in the early phases) is *not supposed to reason in terms of execution*. Since the basic paradigm is that of proof, he has to think directly *in terms of properties* that have to be satisfied by the future system. This shift from execution to properties and proofs constitutes, in fact, a great step, which could sometimes represent an insurmountable difficulty to some persons. In our experience, it is not so much the manipulation of well defined mathematical concepts (sets, relations, functions, numbers, sequences, etc) that poses a serious problem to certain **B** practitioners, it is rather the necessary change of habit consisting in abandoning (for a while) the idea of execution.

Necessity of Re-writing the Requirement Documents.

As a matter of fact, some **B** novices use it as if it were a programming language with the classical concepts of discrete mathematics directly at their disposal. From our point of view, it is a mistake. We must say, however, that people are inclined to do so, since the informal requirements of the system they realize are often also written with execution in mind. Very often indeed, such requirements are already written in the form of a pseudo-implementation describing the future system in terms of data and algorithms acting on them.

This is the reason why, in our opinion, it is almost always indispensable, before engaging in any development with **B**, to spend a significant time (that is, for a large system, several months) to just *rewrite these requirements* in english, say, so as to re-orient them towards the precise statements of the properties of the future system. The natural language statements, the diagrams, or the tables describing these properties in one form or another must be identified, isolated and labeled in order to clearly separate them from the rest of the text (hypertext technology and "literate programming" helps here).

Such properties may concern either the *static* aspect of the system (that is, the permanent properties of its data), or its *dynamic* aspect (that is, the properties expressing how the data are allowed to evolve).