System Modelling and Design
What is this all about?
Revision: 1.1, November 16, 2005

Ken Robinson
May 29, 2006
mailto:k.robinson@unsw.edu.au

Contents

1 What have we been about? 1
   1.1 What is the alternative? 2

2 Industrial Experience 2
   2.1 The Paris Météor Line 2
   2.2 Gemplus smart cards 3
   2.3 Recent Siemens experience 3

3 What do we mean by Formal Methods? 3
   3.1 Testing within traditional engineering disciplines 4
   3.2 Contrast with testing of software 4
   3.3 Advantages of formal development and proof 4
   3.4 Formality is inexorable 4
   3.5 Course Objectives 4
   3.6 Proof, Infallibility and Documentation 5
   3.7 Specifications vs Implementations 5
   3.8 Pedigree of B 5

Objectives of this Lecture

This lecture will attempt to wrap up what has been taught during this course and will highlight what you could have learnt.
Also, some larger industrial context will be presented.

1 What have we been about?

We have presented a mathematical semantics that allows the rigorous specification of system functionality.
Isn’t it strange that something like this has not been presented earlier. In no other engineering discipline is this the case.

1.1 What is the alternative?

To those who argue that such a rigorous, mathematical approach to system specification and design is unnecessary, there is a challenge:

_How else would you describe system behaviour?_

2 Industrial Experience

Industrial application of formal methods was able to be pursued in the early 1990s.

Experience in the development of an ATC


This paper describes the development of an Air Traffic Control Information System by Praxis, an English company. The following statement is made:

Using formal methods helped us to build the right system and helped us to build it right — at no extra cost. Our project shows that using formal methods on real, large-scale projects is not only practicable but beneficial The question software engineers should now be asking about formal methods is not whether to use them, but how best to benefit from them as part of a complete software-engineering approach.

2.1 The Paris Météor Line

In 1998, a new Paris Metro line was opened. This was the Météor line, the world’s first completely automatic train line.

This project was implemented by Matra Transport, now owned by Siemens. _B Method (B)_ was used successfully for the design and implementation of the the control system of the Météor line. See


The distributed control system handled the critical parts of the central control room, the wayside equipment along the track and on the platform, the onboard train control. The Météor system was developed by Matra Transport, now owned by Siemens. The following description can be found in the wikipedia pages:


The Siemens site mention the B-Method in connection with _Safety_ and _Innovation_:


The use of B was mandated by the Paris transit authority, RATP.

The following page talks about the announcement of the award of a contract to install the Météor system on the New York subway.

2.2 Gemplus smart cards

Gemplus smart cards
For another application see


GEMPLUS is a company that develops smart cards.

2.3 Recent Siemens experience

At the recent ZB2005 conference, Frédéric Badeau presented a paper jointly authored with Arbaud Amelot, entitled Using B as a High Level Programming Language in an Industrial Project, Roissy VAL.

This paper describes the use of B on the development, by Siemens, of a driverless shuttle line at Charles de Gaulle airport. Again B is being used to develop the wayside control system for this project.

The paper presents some interesting statistics.

Roissy VAL statistics

<table>
<thead>
<tr>
<th>B model</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract model</td>
<td>28,163</td>
</tr>
<tr>
<td>Automatic Concrete model</td>
<td>117,565</td>
</tr>
</tbody>
</table>

The total number of proof obligations was 43,610, of which 3% (1,441) were discharged interactively.

This is serious formal verification.

An interesting observation

Siemens use B as a programming language and have developed automatic refinement tools to generate B0, from which (Ada) code is produced.

The paper comments:

… it is interesting to note that when a member of the project development team took a look at some ADA code corresponding to an automatic refinement level, this person was lost and could hardly make the link with the software specification.

3 What do we mean by Formal Methods?

In our use of the term Formal Methods we mean the application of mathematics—set theory and logic—to specify, design and implement software—or more generally systems—in such a way that the resulting code has been proved to be consistent with the original specification.

In B, a specification is a mathematical model of the required behaviour of a system. Specifications are generally abstract.

We then transform the specification through a sequence of formally defined refinement steps towards a concrete implementation.

During the process we have a number of proof obligations that must be discharged.

Conventional software development methods usually express the requirements informally, in either structured or unstructured English, or using some structured notation: dataflow, entity relationship diagrams, or the unified modelling language (UML). Specifications are frequently expressed directly in programming code. Design then consists of “fleshing out” the code to produce an implementation.

In contrast, when using a formal development method, like B, the specification is an abstract description of the requirements, expressing what behaviour is required, rather than how to produce that behaviour.
It follows the design phase must affect a radical transformation of the specification in order to obtain executable code.

3.1 Testing within traditional engineering disciplines

If we examine traditional engineering disciplines, such as electrical engineering, or civil engineering, we find the following:

- Designs are based on a mathematical theory of the materials, components, structures to be used in the implementation of bridges, buildings, electronic circuits.
- Then the testing consists of physical testing of the implementation, or a model of the implementation.

This is a successful strategy because the domains can be described by continuous mathematics: if a model conforms for some specific test input, it will also conform for input that is “less than” that input. Thus testing can be conducted for extreme values within domains.

This strategy does not work for discrete valued domains.

3.2 Contrast with testing of software

Software executes over discrete domains, and testing usually consists of probing points within that space. Thus testing can only confirm conformance of behaviour at specific points. Testing is therefore incapable, in general, of demonstrating conformance over the complete application domain.

Thus testing may confirm the presence of bugs, but not their absence.

3.3 Advantages of formal development and proof

Using a formal development method, we build a model using constructs that are described by precise mathematical theories. These models capture the behaviour in a complete application domain.

As we develop our specifications into implementations, the formal method produces proof obligations that basically describe the complete set of tests that confirm that the behaviours of the specification and the design are consistent—more correctly, not inconsistent.

Discharging the proof obligations is thus the counterpart of testing in other engineering disciplines.

A proof validates behaviour in a complete domain, not simply at a single point.

3.4 Formality is inexorable

The increase of the use formality in software development has been continuous, from formal grammars to specify programming language syntax, to the semi-formal application of translator generators in compiler implementation. High-level programming languages themselves are an instance of increased formality, over machine level (assembler) programming in this case.

Everywhere rigour and formality has been used there has been an increase in the reliability of implementations.

There is no reason to believe that this “progress” will not continue.

3.5 Course Objectives

The objective of the course is to get you to think more carefully about the specification, refinement and implementation phases of software development.

The objective of the use of a formal method is to encourage you to think more rigorously—even formally—about specification in particular, and to extend that to other phases of software system development.
3.6 Proof, Infallibility and Documentation

We have already said that B allows us to prove that the implementation is consistent with the specification, but the words *prove* and *proof* have unfortunate connotations of *infallibility.

*This is not intended.*

Nor should the B-Toolkit be regarded as an oracle, despite the fact that it contains a number of theorem provers—really proof assistants.

The B-Toolkit is a documentation tool that allows the whole development to be documented as human readable evidence of the developers efforts to ensure a sound implementation.

3.7 Specifications vs Implementations

In the specification domain we will be dealing with abstract concepts.

*In the implementation domain we need to deal with finite constructs.*

A good example of the difference between these two domains is the infamous Y2K bug.

*Why?*

The Y2k bug is one of a class of bugs that arise because of the following problem:

*Many of the concepts and functions that we wish to implement in our computer systems are unbounded. For example, the concept of date does not have any known bound: the life of our solar system presumably is limited, but we don’t know what it is. In the 1960’s, when this bug was born, the end of the century looked a long way away.*

On the other hand, the physical computer systems on which we have to implement our systems are finite. We have to map our unbounded concepts onto finite physical hardware. This is part of a process called *design*, and sometimes we get it wrong, very wrong!

3.8 Pedigree of B

The author of the *B Method* is Jean-Raymond Abrial and B is built on the foundations established by

Tony Hoare: *Pre and Post conditions*

Edsger Dijkstra: *Weakest Preconditions*

Cliff Jones: *VDM*

Ralph Back, Carroll Morgan: *Refinement Calculus*

Jean-Raymond Abrial: *Z* specification notation and mathematical toolkit

Many others have, of course, made significant contributions to related formal methods.

- **1985** Initial B concepts developed by J-R Abrial
- **1985–1988** BP-funded research project at PRG Oxford and BP IT Division, headed by J-R Abrial
- **1988–92** Commercial development of tool kit (with subcontract to GEC-Alsthom, France)
- **1992–** Alpha tests in industry
- **1994–** Release of toolkits

Many others have, of course, made significant contributions to related formal methods.