System Modelling and Design

Refining Software Engineering

Ken Robinson

Rodin Workshop
Southampton
16th July 2009

Contents

Overview

This talk is going to be presented from the point of view of a user of Event B and Rodin, in particular, the use of both in the teaching of undergraduate software engineering students at UNSW (Sydney Australia)

We are concerned with help students produce designs that they understand.

This talk is about how we try to do that using EventB.

Software Engineering

This talk is largely concerned with *Software Engineering* and software engineering education.

There is an interesting sign as you ascend the stairs in this building:

Dependable Systems & Software Engineering

There is surely more than a suggestion that *Dependable Systems* and *Software Engineering* are mutually exclusive?

To me it expresses a sad truth of software engineering.

System modelling not formal methods

While many people —perhaps most— will regard EventB as a *formal method*, I strongly resist that classification.

My objectives are:

1. to help students reason about their designs;
2. to help them to appreciate design, as distinct from implementation;
3. to show them that there are rigorous ways of understanding systems in general, and software in particular.

Engineering Methods

To do all of the above we will use engineering methods to model and then design Systems.

That will involve the use of mathematics, in this case set theory and logic.

Mathematics is an intrinsic part of all engineering design.

I find the term formal methods to be counter-productive. To begin with the word *formal* tends to obscure the fact that any design process —any engineering process— will involve a significant amount of informality. But beyond that I find that people think that because formal methods involves proof then therefore they are given assurance that whatever they produce is correct.

I want to emphasise the fact that discharging proof obligations in event B gives proof of consistency rather than proof of correctness; indeed I stress the fact that all humanly engineered systems can fail.

Change of Vocabulary

The above involves a change of vocabulary:

verify in place of *prove*

consistent in place of *correct*

The above all goes to reinforce the notion that the correctness of a model depends on the informal interpretation of the requirements, and the subsequent expression of that interpretation in, say Event B.

Courses

I'm going to discuss three courses that students in our software engineering program take in their second year.

Semester	Course
1	System Modelling and Design
1	Software Engineering Workshop 2A
2	Software Engineering Workshop 2B

System Modelling and Design is the course in which I use event B and Rodin to teach modelling and design

The other two courses are software engineering workshop courses in which students work in teams on the design and implementation of some system.

No Concept of Design

I find that the students who take COMP2111, although in many cases being very good to outstanding programmers are unable to discuss the design of their programs.

Indeed they regard a request to discuss the design of a program as being nonsensical and are usually unable to give an answer, let alone a good answer.

Have you ever tried that sort of exercise?

They have been taught to write programs, to think in programs, but never what the program means. The only way they know of verifying a program is to test it, and, of course, they still can't explain their programs.

To me this is a serious flaw in the education of software engineers.

Learning how to model

In this course I try to teach the students how to use non-determinism, abstraction and logic to model systems in which they have confidence in the model of the system's behaviour.

I also encourage the development of a model through a sequence of refinement layers, in which each layer deals with one aspect of behaviour.

Essentially I am following what Abrial calls quote " correct by construction".

Moving to Event B: the experience

I have been using Classical B and Event B since about 1996 and although both forms of B. have essentially the same mathematical toolkit and are essentially interchangeable I have found event B. to be a significantly different experience, and a consistently rewarding experience.

An example

I will illustrate with an example.

I have been using a traffic lights example for a very long time. When I came to recast the example in event B I had an insight which I'd never seen before. My previous examples have had red, green and amber lights with events or operations to change the lights to one of those colours.

This now seemed inappropriate. It seemed more appropriate to consider the steady state of an intersection controlled by traffic lights to consist of only red and green lights. Amber lights only occur during a transition. The amber lights are essentially a safety device to prevent sudden changes from green to red. Thus the top level model of a traffic light system consists of a state consisting of only red and green lights and two events to change the light in a particular direction to green and red, respectively. The refinement introduces amber to the state and provides a number of events to sequence between top level states.

The Software Engineering Workshops

In the workshop that runs in parallel with COMP2111, students work on the specification or model of a system that they will take through to an implemented prototype in the second workshop. The model that the students produce is in event B.

Last year students worked on an iTunes type system. This year students are working on an eBay type system. This year the modelling has gone very well with students demonstrating their models using AnimB. AnimB provides very impressive animation.

In the second workshop students will take their model and produce an implementation of a prototype. At the moment, the implementation is done using informal translation rules to translate into either Java or Scala.

Searching for a model

At the time the students were working on producing their Event B model of their eBay system, I introduced a model of a lift control system into COMP2111 to demonstrate the layering approach on a non-trivial example.

The lift control system model consists of a the following:

1. basic lift system in which the rules for lift movement are established but lift movement is essentially non-deterministic
2. refinement in which we add lift doors
3. refinement in which we added floor doors
4. a refinement in which we add lift buttons
5. a refinement in which we add floor buttons

The first three layers were completed in lectures

The remaining two layers were given as an assignment.

Learning to Love Nondeterminism

Students probably started the courses thinking that nondeterminism was some obscure, esoteric concept.

They hopefully ended the course understanding that nondeterminism is real and it exists in many systems that they would not have thought of as nondeterministic.

Implementing the model

An implementation of a model depends on the context of the system being modelled, so there can be no general implementation strategy for Event B models.

For the implementation of the eBay model —as for the implementation of the iTunes model last year— the following strategy will be followed:

1. the machines will be mapped onto OO classes;
2. the guards of the top-level events will be moved into a GUI to give a set of "buttons" that represent the events, to which the guards were attached. The buttons may be dimmed if the corresponding events are not currently enabled

This seems an appropriate implementation strategy for this type of model.

We will look seriously at the use of Scala (Martin Odersky, Lex Spoon, Bill Venners) for implementation of Event B.

Acknowledgements

I would like to acknowledge

- Deploy through Michael Butler for the invitation to attend this workshop
- Jean-Raymond Abrial, Laurent Voisin, for conversations and assistance in learning to use Rodin.
- Christophe Metayer for his brilliant work on AnimB
- Peter Ho for enabling me to assist students with modelling their systems using Event B. Peter runs that particular software engineering workshop course.

Answers to Questions

The following are answers to questions raised during and after the talk.

1. Lift system: *do we allow students to change the machines/events already defined?* The only changes we expect are those consistent with refinement.

2. Requirements in workshop: *Do we deal with requirements?* Yes, the main presentation doesn't say this, but have a set of requirements consisting of some basic set and extended by their own requirements. Throughout the project they are required to maintain tracing of requirements in both directions. This we regard as critically important as otherwise the EventB model is essentially useless. The requirements are hand-coded, but a plugin would be very nice.

3. *"Top-level" events in implementation* This term was intended to indicate events before refinement. They could be first specified at various levels of the refinement. Such an event could, through refinement, have many different incarnations each with their mutually exclusive set of guards.

4. *Model critique:* the first exercise in the second of the two workshops is for students to view all the models produced by all teams and then to take 3 models plus their own and write a critical evaluation of those 4 models. An outcome of that exercise is for the team to possibly revise their model, adopting ideas from the models of other teams.

5. *Copying in workshop* Copying in the workshop is not a problem. Quite apart from the fact that students are encouraged to adopt ideas from other teams' models, in general the teams are very secretive about their ideas. They are very competitive.

6. *Team size* Our experience is that 4 is the best team size: 3 is a bit too small and 5 allows one student to coast on the efforts of other members of the team.

7. *Programming vs design, logic etc* While many students are good at programming that is not to say that they find it easy. They certainly find the active use of logic challenging. They've all taken a course in discrete math that is largely aimed at students in our computing programs.

8. *Discharging POs* While discharging POs is not easy students seem to like rising to the challenge, and some of them get very good at it. On assignments I give the PO statistics on my solution to the assignment. This acts as a reasonable guide, especially to the question of whether their invariant is too weak.

9. *Assessment* Assessment for COMP2111 consists of assignments; a multiple choice/multiple correct answer exam and a short answer exam. Assessment for SENG2020 consists of presentations, reports and demonstration of their prototype.

10. *Level of students* The students in the above courses are second year undergraduate students in a Software Engineering program.