Teaching Formal Methods for Software Engineering – Ten Principles

Antonio Cerone[1] Markus Roggenbach[2] Bernd-Holger Schlingloff[3] Gerardo Schneider[4] Siraj Ahmed Shaikh[5]

[1]United Nations University — UNU-IIST, Macau SAR China

[2]Swansea University, Wales, United Kingdom

[3]Humboldt University and Fraunhofer FOKUS, Berlin, Germany

[4]University of Gothenburg, Sweden

[5]Coventry University, United Kingdom

Prague, April 7th, 2019

Motivation

Formal Methods are one means in Software Engineering that can help ensure that a computer system meets its requirements.

The use of Formal Methods in Verification and Validation is wide and includes techniques such as static analysis, formal testing, model checking, runtime verification, and theorem proving.

In many engineering-based application areas of computer science, e.g., in the railway domain, Formal Methods have meanwhile reached a level of maturity that already enables the compilation of a so-called body of knowledge.

In Computer Science education, however, Formal Methods often play a minor role only.

Some common questions

Typical questions raised in curriculum discussions include:

- Which of the many Formal Methods shall be taught? Will the material taught be accessible to mainstream students?
- Will a Formal Methods course be attractive to students?
- Are there teachable case studies beyond the toy example level?
- Can students be involved in real projects where they can apply Formal Methods?

This effort

We propose a constructive and positive response to such questions by writing a book “Formal Methods for Software Engineering – Languages, Methods, Application Domains”.

We adopted some principles in the writing of this book.

We also share some experience of teaching the book contents at summer schools associated with the SEFM conference series, and at different universities.

SEFM Summer School 2008 (Cape Town)

SEFM Summer School 2008 (Cape Town) (2)

The Ten Principles

1. The field of Formal Methods is too large to gain encyclopaedic knowledge – choose representatives
2. Formal Methods are more than pure/poor Mathematics – focus on Engineering
3. Formal Methods need tools – make them available
4. Modelling versus programming – work out the differences
5. Tools teach the method – use them
6. Formal Methods need lab classes – create a stable platform
7. Formal Methods are best taught by examples – choose from a domain familiar to the target group
8. Each Formal Method consists of syntax, semantics and algorithms - focus uniformly on these key ingredients
9. Formal Methods have several dimensions – use a taxonomy
10. Formal Methods are fun – shout it out loud!

Principle 1

Principle 1: The field of Formal Methods is too large to gain encyclopaedic knowledge – choose representatives

The variety of Formal Methods is overwhelming. This might leave a beginner lost in the field.

We recommend teaching a non-representative selection of methods with a solid introduction to each of them.

This is feasible given that concepts studied, say, in the context of process algebra, are also to be found in temporal logics, which again are closely connected to automata theory, and are applied, e.g., in testing.

Principle 2

Principle 2: Formal Methods are more than pure/poor Mathematics – focus on Engineering

Over the last several years Formal Methods has become more of an engineering discipline.

Many CS students quit because they cannot see the relevance of their first years' Mathematics to the engineering problems they want to solve.

Thus, we encourage to motivate the introduction of Formal Methods by engineering challenges.

Principle 2 (2)

We see Formal Methods as part of Software Engineering curriculum.

The use of Formal Methods in software development should not be constrained to a specific process or a life cycle model. It can be used with traditional as well as agile models.

Moreover, Formal Methods should not constitute separate phases, but should rather be integrated as part of the general validation activities. Thus, teaching Formal Methods should frequently resort to other topics in Software Engineering.

Principle 3

Principle 3: Formal Methods need tools – make them available

Formal Methods need tool support in order to become applicable for demonstrating convincing case studies.

Tools for simulation of behaviour and visualisation of state space or traces are essential to allow students to understand the behaviour associated with their models.

Moreover, software systems are fundamentally different to mathematical theories.

Principle 3 (2)

Number of axioms
: Number of axioms when applying Formal Methods is by magnitude much larger than those involved in mathematical theories.

Ownership and interest
: Software code is often more restricted than mathematical theories. Therefore, proofs related to software are studied by few. Consequently, tools play the role of "proof checkers" for quality control in Formal Methods.

Change of axiomatic basis
: Software requirements change frequenty. Design steps involving Formal Methods often need to be repeated several times. Mathematical theories, however, are much more stable. Consequently, tools are needed to help manage such change.

Principles 4 and 5

...please read the paper...

Principle 6

Principle 6: Formal Methods need lab classes – create a stable platform

Carefully designed lab classes can be enormously motivating for students

Apart from offering hands-on experience, lab classess provide students with a sense of achievement: students use Formal Methods with a concrete, visible effect on their screen – rather than being lost or even frustrated in some semantical detail.

To minimise teaching effort, we propose working with frozen versions of tools compiled on a live-CD.

A live-CD helps circumvent any installation problems (which do occur!).

Principle 6 (2)

Demonstrating the Uniform Candy Distribution puzzle.

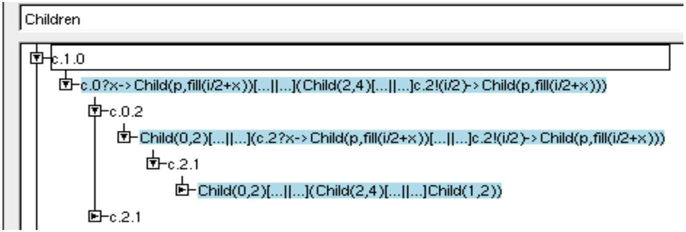
[Lab sheet for the process algebra CSP]

1. Entering the command `probe` starts the simulator ProBe from a terminal. The simulator opens in a GUI.
2. One loads the file `childrensPuzzle.csp` by choosing the File option. This launches a new window.
3. Double clicking on the process `Children` allows one to expand the process as shown in the lectures.
4. Following any execution branch eventually yields a state where all three children hold 4 candies.

In the lab students explore and experience that (1) the system has many different execution paths and that (2) these paths are infinite.

Principle 6 (3)

Once the students have tried the tool, we can then change the example (for example change the number of children, i.e., processes, involved in the puzzle). Only then students should be asked to use their own examples.



```
Children
c.1.0
  c.0?x-> Child(p,fill(i/2+x))[...||...](Child(2,4)[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
    c.0.2
      Child(0,2)[...||...](c.2?x-> Child(p,fill(i/2+x))[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
        c.2.1
          Child(0,2)[...||...](Child(2,4)[...||...]Child(1,2))
    c.2.1
```

Such an approach separates learning the (basic) functions of the tool, dealing with mistakes and error messages (when changing the example), and mastering the method (when working out an own example from scratch).

Principles 7, 8 and 9

...please read the paper...

Principle 10

Principle 10: Formal Methods are fun – shout it out loud!

Psychology tells us that the human learning capacity is highest when we enjoy what we are doing.

Reflections

We put our principles to test by teaching the material at three international summer schools.

We have also used the material in several classes at our home institutions.

Some feedback from the students include

- “The school opened up new ways of thinking.”
- (The school helped in) “showing that theory comes from practice and vice versa, but practice is the goal.”
- (I liked) “the practical applications of Formal Methods.”
- (The required) “hands-on exercises were very good.”

SEFM Summer School 2012 (Thessaloniki)

SEFM Summer School 2012 (Thessaloniki) (2)

Reflections (2)

Even though we dedicated quite a large portion (around one third) of the available time to lab classes, according to the participants they could have played an even bigger role. Some “Suggestions for Improvements” included

- “More practical sessions with the tools.”
- “More on applications, especially security.”
- “More lab lessons.”

There is good feedback, and importantly, appreciation for practical hands-on sessions.

What next?

The challenge of Software Engineering has been acknowledged for decades now; the notion of software crisis has been related to the debate on software industry since early days.

Safe to presume the state of the industry will continue this course unless methods and practices are addressed.

Our work is motivated by the very challenge.

The book is due out. Watch this space!

Thank You