

Taking command of software design

Dr Eric Hehner entered the world of computing shortly after the discipline emerged. With over 40 years of experience, he has helped shape the formal methods underpinning in programming evolution



Who has been influencing and supporting your research at the University of Toronto?

In 1977 I was fortunate to be invited to join the International Federation for Information Processing (IFIP) Working Group 2.3, a team of about 30 people from all over the world who collectively invented formal methods, winning all the top awards in computer science, including six Turing Awards. We meet for one week every nine months or so and discuss our shared research. They have all been an influence on me, and perhaps I have been an influence on them. I can't list them all, but I will mention just two. Edsger Dijkstra was both an inspiration and a good friend; Sir Tony Hoare still is an inspiration and good friend.

Programming is a useful skill to have, with the number of people teaching themselves growing daily. What advice would you give to these individuals to improve their theory and practice?

People teach themselves to throw a ball, but that doesn't make them ballistics experts. People teach themselves to strum a guitar, but that doesn't make them musicians (though I admit that a lot of people make

a lot of money playing guitars badly). And people teach themselves to program (as I did), but that doesn't make them software engineers (though a lot of people make a lot of money writing programs badly).

When I wrote my first programs, I had no idea that writing a program could be as reliable as proving a mathematical theorem. That's what you learn from a formal methods course. I am not saying anything against teaching yourself to throw a ball, strum a guitar or program; they are all fun. But if you want to advance to the next level, you need a course. As it happens, I offer an online course that is free, you can start anytime, and proceed at your own pace (www.cs.utoronto.ca/~hehner/FMSD).

Among your achievements, you published a *Practical Theory of Programming* in 1993. How have you revised the book to keep the information up to date?

1993 was a couple of years before the internet became widely available. By 2002 I had accumulated a list of improvements and updates and some new material, and it was time for a second edition. By then, the internet was well established, and I wanted to make my book freely available on the internet. Springer, who owned the copyright, refused. How can they and I make money if our product is freely available?

To an author of an advanced-level book, the money is not significant; it can never repay the work of writing the book. I didn't give up the fight, and I had some powerful allies, so in the end Springer allowed me to put the book on the web. From my point of view, the main benefit was the ability to make changes. When I discovered how probabilistic programming could benefit from my formal methods, I added a section. Anytime I saw a way to improve an explanation, or to shorten a proof, I made the change that same day. I always want my book to be the best I can make it

today, not just the best I could make it 10 years ago. I stopped calling editions 'first', 'second' and so on, and started calling them by year-month-day. I also keep a change log, available for all to see.

Formal methods you have helped create could one day become the industry norm. Could you highlight some of the real-world applications seen to date?

Formal methods, not just mine, have been used to develop and verify telephone switching systems and internet communication protocols, as well as aeroplane cabin communications. They have to be used for safety-critical software such as medical systems, nuclear power plant controls and aircraft attitude monitors. The largest software that uses formal methods is a compiler, and the largest hardware is a processor (CPU). More commonly, it is used for parts of systems, like the kernel of a secure distributed operating system (software), and a floating-point unit (hardware). Formal methods were used to develop Paris' automated (driverless) metro, and China's railway controls. BMW uses formal methods to develop its monitoring and reaction systems. But formal methods are not yet used for banking and financial software, nor for most application software.

What does the future hold for quantum computing?

According to the principles of quantum physics, quantum computers should be able to perform computations that are infeasible on today's computers. I am no expert at building quantum computers. Those that have been built have very few qubits (quantum bits) of memory, and there are serious problems with decoherence (loss of quantum properties) happening after only a few seconds of operation. I don't know when quantum computers will become practical, but when they do, we have the formal methods for programming them reliably.

{ 00.1v } // // // // // // // //



101110011



Formal **methods** of software design

The formal methods group in the Department of Computer Science at the **University of Toronto** has been working to provide a mathematical foundation for software engineering, helping engineers write precise specifications to say what the software will be for, and then design software whose executions provably satisfy the specifications

DESIGNING ERROR-FREE software is difficult, but it is essential for software on which lives depend, such as aircraft control or pacemaker software. The correctness of each step in the design and development has to be proven in the same way that mathematical theorems are proven. At present, it is possible to develop small and medium-sized programs and the critical parts of large programs this way. In the future, scaling up the use of formal methods to large software projects requires the support of a software development tool that includes an automated prover.

NETTY

Dr Eric Hehner, together with his student Lev Naiman and former students Anya Taffliovich and Robert Will, all at the formal methods group in the Department of Computer Science at the University of Toronto, Canada, are designing and implementing a software development tool named Netty, after Netty van Gasteren, a pioneer in calculational proof. For a long time there have been compilers that tell when a syntax error is made, telling exactly what the error is and exactly where it occurs. Netty takes the next step: it is a logic checker that tells when a logic error is made, and tells exactly what it is and where it occurs. To build such tools and to use them effectively requires knowledge of formal methods. The word 'formal' refers to the use of formal languages for specification and proof so that the entire software development process, including proof of correctness, is at least machine-checkable, and to a large degree machine-generable.

One of the features of Netty is that it keeps track of all specifications and implementations that it has been used for. Then, when anyone uses it again for a task, or a part of a task, that is the same as or sufficiently similar to one that it has already been used for, Netty completes the software development automatically.

HOW IT WORKS

Hehner's main innovation is to treat programs the same way as specifications, so that pieces of program and pieces of specification can be freely mixed and connected. Specifications can employ programming notations whenever they are helpful, and engineers can reason about computations in the full logic, using both the logic notations and the programming notations. The reason this is valuable is that engineers start with a specification, end with a program, and in the middle of software development they have a meaningful mixture. During development they are privy to whether the development is correct; they do not have to wait until the end to find bugs. The same formal methods that tell when a mistake is made during software development also tell when a mistake is made during software modification, and that is a big source of bugs.

Hehner's theory is described in his book *Practical Theory of Programming* (first edition Springer 1993, current edition free online at www.cs.utoronto.ca/~hehner/aPToP). There were previous theories, starting in 1969 with Hoare Logic, which uses a pair of predicates for specification. Then came Dijkstra's weakest precondition predicate transformers in 1976, and others since then, such as VDM, Z and B.

Hehner's theory is simpler, using a single Boolean expression for specifications. The theory is also more general, applying to both terminating and nonterminating computation, sequential and parallel computation, stand-alone and interactive computation. It also includes time bounds, both for algorithm classification and for tightly constrained real-time applications. A US Government report noted: "In the most advanced manifestation, formulated by Eric Hehner, programming is identified with mathematical logic. Although it remains to be seen whether this degree of mathematization will eventually become common practice, the history of engineering analysis suggests that this outcome is likely".

OTHER APPLICATIONS

Along with former student Professor Theo Norvell, now at Memorial University, Hehner has applied the same theory to the automation of the logical aspects of digital circuit design, making it possible to design large-scale circuits that are entirely verified. To design a circuit, one

The team

The formal methods group at the University of Toronto consists of Professors Eric Hehner, Marsha Chechik, Azadeh Farzan and their students. Hehner develops formal methods as an aid to software design and modification. Chechik and Farzan develop formal methods to verify the correctness of, or find bugs in, existing software.








Summer school on formal methods, Santa Cruz, California 1979. Clockwise from left to right: Edsger Dijkstra (then Professor, Technical University of Eindhoven, The Netherlands), student, John Backus (Fellow at IBM, inventor of Fortran, co-inventor of Algol in 1958 and inventor of BNF, which is used universally to describe the syntax of programming languages), student, Dr Eric Hehner (red beard), Professor Rod Burstall (University of Edinburgh, UK).

can just write a program that would perform the same function. They provide a scheme for translating the program to a circuit, and also a proof that their translation scheme is correct. The resulting circuits are smaller and faster than the usual synchronous circuits due to the absence of a global clock. They are also smaller and faster than asynchronous circuits due to the absence of 'handshaking', which is how asynchronous circuits (those without a clock) determine what should happen next. These gains are achieved by calculated local delays that are a by-product of the proof.

Hehner has also applied his formal methods to probabilistic programming, and solved a long-standing open problem known as 'the two envelope problem'. Problems about probabilities usually describe some events or activities; there may be a sequence of them, or some parallel activities or events; there may be some that are conditional upon others. Describing such situations formally is exactly what programs are for. So a program is written to describe the situation, but it is not executed. Instead, the programming notations are interpreted as probability distributions. In other words, the program already expresses the probabilities, it just has to be simplified, like any arithmetic calculation. In fact, a probability calculator could be used, and Hehner's team has built one. He adds: "I don't actually care about the two-envelope problem, although probabilists and philosophers

Terminology

-  **Boolean expression:** an expression that has two possible values
-  **Predicate:** a parameterised boolean expression
-  **Specification:** a description of the purpose of a program
-  **Formal specification:** a specification written in a formal language that the computer can read
-  **Formal methods:** using formal specifications for developing software or checking the correctness of software, preferably with the aid of an automated tool

apparently do; I just wanted to illustrate how my probability calculations work".

STUDENT SUCCESS STORIES

Former student Anya Tafilovich, now a lecturer at the University of Toronto, used probabilistic programming to provide a basis for understanding quantum programming (programming a quantum computer). Quantum programs are notoriously unintuitive; the only way to ensure correctness is automated proof. She now has the proof techniques, and is currently automating them.

Meanwhile, another former student Ioannis Kassios, now at ETH Zürich, applied the same formal methods to the constructs found in object-orientated programming languages, like Java. Some of his work has been included in Microsoft's verifier.

Hehner and his students have been laying a foundation for programming and building the tools that will enable software engineers to write more reliable software, hardware engineers to create better digital circuits, and engineers of the future to program quantum computers.

INTELLIGENCE

FORMAL METHODS OF SOFTWARE DESIGN

OBJECTIVE

To make error-free software by applying the methods of mathematical proof to each step in software design. This requires showing what mathematics applies, and how it applies, and also building software tools to aid practicing software engineers.

KEY COLLABORATORS

Dr Ioannis Kassios, ETH Zürich, Switzerland

Dr Albert Lai, independent, Toronto, Canada

Mr Lev Naiman, graduate student, University of Toronto, Canada

Dr Anya Tafilovich, Lecturer, University of Toronto, Canada

Mr Robert Will, Immobilien Scout, Berlin

CONTACT

Dr Eric Hehner
Principal Investigator

Department of Computer science
University of Toronto
Ontario
M5S 3G4
Canada

T +1 416 509 2762
E hehner@cs.utoronto.ca

ERIC HEHNER received degrees in Mathematics and Physics at Carleton University, and a PhD in Computer Science from the University of Toronto. He then joined the faculty, becoming Full Professor in 1983 and Bell University Chair in Software Engineering in 2001. Hehner's research has mainly focused on formal programming methods. He was Visiting Scientist at Xerox Research Center, Palo Alto, USA, Visiting Fellow at Oxford University, UK, Visiting Researcher at the University of Texas, USA, Professeur Invité at the Université de Grenoble, France, Visiting Professor at the University of British Columbia, Canada, and at the University of Southampton, UK. He is a member of the International Federation for Information Processing (IFIP) Working Group 2.1 on Algorithmic Languages and Calculi, and Working Group 2.3 on Programming Methodology. Hehner is an editor of *Acta Informatica* and of *Formal Aspects of Computing*, and has written two books (*the Logic of Programming*, and a *Practical Theory of Programming*), many journal and numerous conference papers.