Software Quality

The introduction of fast and cheap computer and networking hardware enables the spread of software. Software, in a nutshell, represents an unprecedented ability to channel creativity and innovation. The joyful act to simply write computer programs for the existing ICT infrastructure is to change the world. We are witnessing how are lives are changing rapidly as a result, on all levels of organization and society and in practically every aspect of the human condition: work, play, love and war.

The act of writing software does not imply an understanding of the resulting creation. We are surprised by failing software due to bugs, by the inability of rigid computer systems to "just do what we want", by the loss of privacy and information security, and, last but not least, by the million euro software project failures in the public sector. These surprises are generally not due to negligence or unethical behavior. No, they are due to our incomplete understanding. We do not yet fully understand what we are creating. It is, as of yet, all much too complex. Lack of understanding leads to lack of control.

Like it is easy to write a new kitchen recipe for a dish the world has never seen before, it is easy to create a unique computer program, which does something the world has never seen before. Like we cannot easily predict up front how well the dish will taste, we cannot easily predict how well the program will behave. The emergent properties of software are on all levels of abstraction. Three examples: first a "while loop" can be written in a minute, and it can take either a week to a lifetime to understand whether it will terminate eventually or not on any input. Now let's try and plan the budget for a software project for which all loops should be quick to terminate. Second, simply scaling a computer system from a single database with a single front-end application to a shared database with two front-ends applications running in parallel can introduce the wildest unpredictable behavior. The "improvement" may lead to random people not getting their goods delivered, or worse; their other limb amputated. Third, we do not know how the network will react to the load generated by the break of the next international soccer match between France and Germany. When will it all come down to crash?

Higher quality software is simpler software with more predictable properties. Without limiting the endless possibilities of software, we need to be able to know what we are creating. Teaching state-of-the-art software engineering theory and skills is one of the means to make sure we understand better, but this is not enough. We are working on better theories and working on better tools to learn how to understand complex software and how to control its complex emergent behaviors. We will be able to adapt existing software to satisfy new requirements and we will know how costly this will be and what the quality of the result will be. We will design software in a way that consciously made design decisions will lead to predictable high quality software artifacts. We will be able to plan and budget software projects within reasonable margins of error.

The current ERCIM news issue in front of you contains recent steps on the way to software quality. We admit we do not fully understand or control software yet, but we are working on it. Some researchers study the reality of software as it is now, discovering theories and tools to analyze, explain and manipulate. Other researchers are re-thinking and re-shaping the future of software by discovering new simpler languages and tools to construct the next generation of software. The two perspectives should leapfrog us into a future where we understand it all.

As quality and simplicity are highly subjective concepts our best bet is to make software technology more and more contextual. General theory, languages and tools have resulted in overly complex systems, so more specialized tools and techniques for groups of people and industries are now being discovered. For example, instead of modeling computation in general we are now modeling big data processing. Instead of inventing new general purpose programming languages we are now inventing domain specific formalisms. Instead of reverse engineering all knowledge from source code we are now extracting domain specific viewpoints .

We hope you will find this selection of articles and inspiring overview of the state-of-the-art and beyond in software quality engineering research.

Guest editors

Anthony Cleve (University of Namur)
Jurgen Vinju (Centrum Wiskunde & Informatica and TU Eindhoven)