NORTH-HOLLAND

Guest Editor's Corner
Research Directions in Formal Methods Technology Transfer

Hossein Saiedian

Department of Computer Science, University of Nebraska at Omaha, Omaha, Nebraska 68182-0500

Welcome to the Special Issue of *Journal of Systems and Software* (*JSS*) on Formal Methods Technology Transfer. It is certainly an honor for me to present this special issue to the software engineering community and the *JSS* readership.

Formal methods represent one of the most innovative research areas that can significantly contribute to an engineering discipline for software systems. The existence of many international workshops and conferences and many books and special issues of journals devoted to formal methods in recent years bears witness to the importance of this area. Many of the research articles, from both academia and government and industrial research labs, have reported the potential impacts of these methods and how they can substantially increase the reliability of software systems, especially those operating in critical environments.

Formal methods refer to the use of mathematical techniques in the development of software and hardware systems. The focus here is of course software systems. Formal method notations rely on a formal syntax and formal semantics to unambiguously define the functionality of a software system and allow the expected behavior of such a system to be predicted from a mathematical model of that system. Most formal methods notations are based on discrete mathematics. Since software systems are discrete systems (i.e., their behavior can be viewed as a succession of discrete state changes), they can, theoretically speaking, be best described by means of such methods.

Address correspondence to Dr. Hossein Saiedian, University of Nebraska at Omaha, Department of Computer Science, Omaha, Nebraska 68182-0500 USA. E-mail: hossein@cs.unomaha.edu.

The standard practice used by the software practitioners in the software community to describe the requirements of a software system is to use a combination of natural language and diagrams and to employ a manual process to review and verify them informally. Informal verification is subjective and often non-repeatable. Furthermore, the inductive nature of informal verification and analysis runs the risk that some properties and/or scenarios will not be covered or investigated. Such an approach is not very adequate, especially for critical software systems. Since many of the problems in software development, particularly during the requirements specification, are caused by impreciseness, incompleteness, and ambiguity, formal methods can assist most effectively in minimizing such problems. Despite such potential for precise definition of the requirements and for substantial elimination of residual errors in the specifications, formal methods are not widely adopted in the industry. (I must add that there are many cases of successful application of formal methods in industry, many of which have already been reported, see for example the collection entitled *Applications of Formal Methods* [Hinchey and Bowen, 1995]. A second volume of these industrial reports is already planned. However, these cases studies are still quite isolated.)

Given the historical concerns for reliable software, most serious software practitioners in the software building world would welcome new approaches to reliable software construction and therefore would adopt any tool and notation that work toward achieving reliability goals in a cost-effective manner. For various reasons, however, industrial practitioners have been reluctant to consider formal methods very widely despite the flurry of research results suggesting the applicability and effectiveness of these

0164-1212/98/$19.00
PII S0164-1212(97)00164-7

methods. This is perhaps because industrial practitioners view the sometimes deep analysis of formal methods more as speculation than practical evaluative research. This is partly due to the fact many industrial practitioners have not studied modern software engineering (or even computer science) at university.

The purpose of this special issue of *Journal of Systems and Software* is to address this very issue, that is, why, in spite of many research results asserting the practical applications of formal methods for increased reliability, we do not see wide usage. Our objective is to explore ways in which the benefits of formal methods—whatever constitutes the most important benefits of formal methods—can be transitioned into practice and how the gap between the expectations of industrial practitioners and the research results of academia can be narrowed. To achieve this goal, we invited articles that presented:

- initiatives to narrow the chasm between practitioners and researchers,
- empirical results in applying formal methods to large systems,
- evaluative explorations of the costs and benefits of formal methods,
- integration of formal methods with non-formal ones,
- formal methods light and partial applications,
- strategies for formal methods that can scale to large systems,
- initiatives intended to increase and improve practitioners' interests and confidence in formal methods, and
- initiatives intended to increase and improve researchers' understanding of the role of formal methods in large systems.

In line with the aims and mission of *Journal of Systems and Software* (see Editor's Corner, *JSS* 28:1–2, 1995), submissions that reported on deep and theoretical analysis of formal methods where the conclusions of those analyses were not supported by some form of evaluation or did not present substantial issues were excluded or were given a low priority.

As you will notice, we considered the opinions of those who had adopted formal methods and who have had exposure to such methods as well as opinions of those who are skeptical. Accepted articles represent research carried out in universities, private organizations, and government laboratories from both Europe and North America and include diverse views, those that focus on the impact of formal methods on software practice and strategies for furthering such impact in the future as well as those that are critical of formal methods. This was in the hope of providing a forum for debating issues that were very related. A short summary of each article follows.

The first two articles are invited articles by two internationally renowned individuals, namely, Michael Jackson and David Parnes, both of whom have made major contributions to the software engineering and computing science fields. Michael Jackson investigates a number of very important issues related to the transfer of formal methods technology by looking at traditional engineering and highlighting what traditional engineers do. David Parnas suggests that for the formal technology transfer to succeed two things must be done: (1) integrated formal methods into the programming and software course and (2) improve the methods until they are consumeable by the practitioners.

Steve Easterbrook and John Callahan (NASA/WVU Software Research Lab) describe a case study of the lightweight use of formal methods for verification and validation of portions of a large, natural language specification. Their study arose from a need within the project to analyze a set of detailed requirements that could not be verified using manual techniques. The requirements were restated in a precise, tabular form, and two methods, SCR and SPIN, were used to test different properties of the same subsystem. A number of defects were discovered in this way, leading to an improvement in the quality of the original specification. The study demonstrates that lightweight formal methods provide a useful tool for debugging specifications, even where they do not guarantee correctness. It also demonstrates that an independent verification and validation team can apply formal methods as an analytical tool, even where the developers do not produce any formal specifications themselves.

Jim Armstrong (British Aerospace Dependable Computing Centre) discussed an approach to formal methods technology exploitation which combines graphical formalisms with traditional theorem proving techniques. The link between a graphical formalism and a theorem prover can be provided by means of an "omega" function, which provides a axiomatic semantics for a subset of the graphical language. This function can be used to generate formal representations of the specification that are either suitable for a prover, or capture different "views" bringing into relief specific types of information. The example omega function in the article maps a subset of statecharts into Real Time Logic, and explicitly includes timing information. The difficult issue of ensuring that the formal representation of the graphical specification is sound with respect to tool support is addressed through a combination of subsetting and formal verification. The author describes how the approach has developed since its inception, stressing the importance of basing a formal notation upon appropriate abstractions, and of alignment with a tool-supported graphical language.

Today's telephone systems—the so-called Intelligent Networks (IN)—offer the possibility to modify their behavior by activation or deactivation of different IN services. Since these services may access the same resources or may have conflicting aims or even may be implemented incorrectly, services in an IN can interact in an undesired way. This fact is known as the service interaction problem.

Ulrich Nitsche's article (University of Zurich) deals with tackling the service interaction problem using formal verification techniques and behavior abstraction. In a first step, the specification of a service is checked separately for its basic properties. In a second step, this specification is embedded into a specification of other services which are checked for interaction with the considered service. The behavior of the service is extracted from the behavior of the combined services by an abstraction step. If the extracted behavior still satisfies the services' basic properties under assumptions discussed in the article, the other services do not interact unintendedly with the considered service. Otherwise, a service interaction is detected. In the article, the presented techniques are used to show an interaction of the services *Call Forwarding Unconditional* and *Selective Call Rejection.*

Lalita Jagadeesan (Bell Laboratories) and her industrial and academic colleagues claim that the development of formal methods has outpaced their use. Although there are theories that try to explain this situation, many of them are overly simplistic. They state that technology transfer is a complicated process involving many groups of people with many different goals and objectives. They discuss their experiences in trying to introduce a specification-based testing method into a commercial product. A two-step technology-transfer process is used: feasibility study and then usability study. The feasibility study explores the effectiveness of the tool in a laboratory setting. The usability study explores the effectiveness of the tool in a commercial setting. Their work finds that each study uncovered different types of problems and played an important role in the technology transfer.

Sara Jones (University of Hertfordshire), David Till (City University), and Ann Wrightson (University of Huddersfield) report on an international workshop, held at City University, London, UK in December 1996, whose aim was to consider the interaction points between requirements engineering and formal methods. The workshop was organized jointly by two special interest groups, one concerned with requirements engineering and the other with formal aspects of computing. Invited speakers from the UK and abroad addressed a series of selected issues from the point of view of the requirements engineer and from the point of view of those who have developed relevant formal methods and tools; there were also sessions devoted to more general discussion. They begin by looking at what has already been achieved in this field. The article then draws on the presentations of the invited speakers, and on the discussions which took place, in order to bring out what are currently seen as the most important challenges to be addressed and the areas where the most productive synergies could be achieved.

Finally, Baudouin Le Charlier (University of Namur) and Pierre Flener (Bilkent University) claim that requirement specifications are necessarily informal. They state that the very reason that the running of a program is useful, namely that its results can be straightforwardly interpreted as a statement about the real world, can be used to conclude that the specification of a program only consists of (the statement of) the link relating the program (formality) and its purpose (informality). Since this purpose must be directly understandable, specifications also are the essential tool for constructing, in practice, correct real-world programs through explicit but non-formal reasonings. They explain why formal specifications are not really specifications, since this would be a contradiction in terms. They agree, however, with the proponents of formal methods on most of their arguments, except that specifications should be written in a formal language, and, inevitably, on the consequences of the assumption.

I hope you enjoy and benefit from this special issue as much as I enjoyed the process.

REFERENCE

Hinchey, M. G. and Bowen, J. P., editors. *Applications of Formal Methods*. Prentice-Hall International, 1995.