Providing Trusted Components to the Industry

Bertrand Meyer, EiffelSoft
Christine Mingins and Heinz Schmidt, Monash University

The software industry stands on feet of clay. However carefully we may strive to build correct and reliable software, we have no way of guaranteeing the quality of the result. Building correct and reliable software depends on the quality of so much else, from the hardware and the operating system to the compiler and the runtime libraries. And any significant software system has so many details and components of its own that we can hardly expect to get everything right if we do it all by ourselves.

None of the commonly suggested approaches suffices to quell these worries: "Test, test, and retest," the implicit motto of much of the industry, is expensive and wasteful. Yet as any user of PC software knows all too well, the result of all this sweat is far from ideal. Software companies systematically and openly ship software that has a large number of known bugs.

On the academic side, much progress has been made toward formal methods, through which it is possible (in principle) to *prove* the correctness of program elements using mathematical techniques. But results from this important field have had only limited effect on the practice of the industry for several reasons:

Editor: Bertrand Meyer, EiffelSoft, ISE Bldg., 2nd Fl., 270 Storke Rd., Goleta, CA 93117; voice (805) 685-6869; ot-column@eiffel.com

The goal of this project is to provide the entire software industry with a powerful set of components deserving a high degree of trust.

- It is too difficult to build mathematical models of the most delicate aspects of "real" programs, from floating-point computation to pointers.
- There are few powerful tools to assist this effort.
- Even more fundamentally, it is just too expensive and difficult to apply formal techniques thoroughly.

The recent push for reuse and componentware has raised the hope that by relying on reusable components we can gain quality and reliability. But without excellent techniques to build the components themselves, this nirvana is a mirage. In fact, the spread of less-than-optimal components could lead to a *worsening* of the situation.

At least one major industrial disaster—to the tune of half a billion US dollars—has already been ascribed to the reuse of an improperly specified component. With the progress of incompletely designed approaches to reuse, more catastrophes are likely to happen, leading to a broad rejection of the very idea of reusability, unless the industry takes measures to guarantee component quality.

Formal methods, because of their cost, only make sense when applied to widely reused components, which can recoup the investment through effects of scale. (Possible exceptions are the mission-critical systems for which the stakes are so high that money is not the issue.)

Widespread reuse is only attractive if we can be formal enough to guarantee that the components will be correct, aiding rather than harming the systems that will rely on them. (See "The Next Software Breakthrough," *Computer*, July 1997.)

In this column, we describe an ambitious but realistic project that combines the ideas of reuse and formality with other more pragmatic techniques. The goal is to provide the entire software industry with a powerful set of reusable components deserving a high degree of trust.

HOW TO ENSURE TRUST

No single technique can produce completely trusted components. Trust is in fact a social phenomenon. Even in mathematics, the most formal of all disciplines, professionals only *believe* in a theorem based on a mix of formal criteria, such as published proofs, and social ones, such as

- who produced the theorem and its proof,
- where it was published,
- who reviewed the publication,
- who else already believes it,
- how much the result has already been applied,
- whether it is consistent with other results in the same area or others,
- whether it gives the "right feeling," and
- whether the theorem and proof are "elegant."

Part of the reason is that almost all published proofs omit intermediate steps to

avoid overwhelming the reader with useless complexity. The complexity in this case is not unlike the complexity of a software system, except that many industrial systems are far more complex than the average mathematical proof.

As a result, trust—especially in software—will not be a binary proposition: "blindly trusted" versus "untrusted." We may trust Microsoft Word enough to use it for our next paper, but we would not bet a year's salary on the assumption that it will not crash while we are writing the paper.

A striking example of the power of social processes exists in software: the success of free source code, notably the tools developed for GNU and Linux. These tools, some quite ambitious, have been developed by volunteers under the international scrutiny of a network of enthusiasts who, relying on all the tools of the Internet, scrutinize the source code of every new version, test it, report defects, and suggest improvements. This process leads to collective work of unprecedented scope outside of any formal organization or commercial framework.

Some of the results are of astounding quality, leading many commercial companies to select, for example, the free GCC compiler over commercial C/C++ compilers, or Linux—initially the work of a student—over commercial versions of Unix, which are the result of tens of thousands of collective years of development in industry hotbeds. Such examples illustrate the power of this recent phenomenon: volunteer scrutiny as a form of free, global quality assurance.

BUILDING TRUST

The Trusted Component Project proposes to apply a mix of formal and informal approaches. It rests on six principal techniques:

Design by contract. This approach to software construction is meant to ensure software is reliable from the start, by building it as a collection of elements that cooperate on the basis of precise definitions of mutual obligations—contracts.

Formal validation. Use modern techniques and tools such as B or Object-Z in connection with the principles of design by contract.

OO and reuse techniques. Thoroughly apply object-oriented techniques and the strict principles of reusable library design.

Global public scrutiny. Make the components freely available in source form; seek contributions as well as criticism from the worldwide Internet community.

Extensive testing. Take advantage of design by contract and focus on component reuse.

Metrics efforts. Track component properties in a controlled fashion.

None of these ideas by itself will do the job. But by combining technical and social processes, we can hope to build a set of components the industry can really trust.

TASKS

The Trusted Components Project has its initial home—in collaboration with Interactive Software Engineering (EiffelSoft)—at Monash University. But it is beyond the scope of any single institution and can only succeed as a long-term collaborative project between many organizations in academia and industry.

The results will be of many kinds—publications and standards as well as trusted software components. The major areas of effort include:

- Choosing areas for component development. Starting with the most humble areas, we can replace the feet of clay with more solid material.
- Developing base components. The base versions will be developed in Eiffel, which has the proper support for design by contract. The Eiffel Kernel Library can serve as a starting point.
- Adapting language-specific components. Versions of the components will be needed by other widely used languages, such as C, C++, and Java. Interface versions will have to be produced in IDL and Microsoft COM.
- Adapting verification technology. Existing tools (such as those for B) may be appropriate, but both the tools and the techniques will need to be adapted to the proof of reusable components.
- Developing testing technology for reusable components, including standards describing the test cases and test procedures.
- Developing assertion language and other conceptual tools for proving practical components.
- Developing and applying effort metrics.
- Developing reuse-based teaching curricula and applying them to actual courses.
- Identifying interesting application areas and developing application-specific libraries on the same principles as the general-purpose libraries.
- Identifying further techniques and tools for building quality componentware.

There are many ways to get involved. (A Web page is forthcoming at http://www.trusted-components.org; you can subscribe to a mailing list by writing to trusted-request@eiffel.com.) This list includes practical tools as well as theoretical themes; research projects for universities as well as product development by companies; possible contributions by institutions as well as dedicated individuals. We hope that it will operate as a truly cooperative endeavor in the tradition of the Internet.

The Trusted Objects Project offers the prospect of a joint effort that may have a major impact on the evolution of the software industry. We view it as a collective, international effort and hope that many people will be interested in joining it. The aim—providing a solid foundation for the software industry—is worth it. ❖

Bertrand Meyer is editor of the Object Technology department.

Christine Mingins is a senior lecturer and associate head of school at Monash University, Melbourne, Australia. Contact her at cmingins@insect.sd.monash.edu.au.

Heinz Schmidt is associate dean of research of the Faculty of Information Technology and head of software engineering at Monash University. Contact him at heinz.schmidt@fcit.monash.edu.au.