SOFTWARE ENGINEERING NOTES 19, 4 (OCT. 1994), 21-22.

Special Feature: Component-Based Software Using RESOLVE

Large-project considerations usually motivate adhering to exacting software engineering techniques. However, for such techniques to be effective in-the-large, the key concerns such as reusability, verifiability, efficiency, adaptability, composability, comprehensibility, interoperability, maintainability, etc., must have been addressed in-the-small — for the *components* of a system.

Producing quality designs for software components, like component design in more mature areas of engineering, demands simultaneous attention to a wide range of substantive technical problems. The RESOLVE effort is a comprehensive venture with precisely this objective. Its goal is to develop and explore a conceptually robust framework, language, and discipline for component engineering. This special feature on component-based software using RESOLVE provides a digest of published results to date and adds significant new background material.

1. Overview of the Special Feature Papers

The accompanying papers are organized as follows:

I: The RESOLVE Framework and Discipline — A Research Synopsis

II: Specifying Components in RESOLVE

III: Implementing Components in RESOLVE

IV: RESOLVE Components in Ada and C++

V: Annotated Bibliography of RESOLVE Research

We begin in Part I with a summary of some of the previous results of the RESOLVE work. This paper highlights the framework and discipline; other publications are available to provide details for the serious reader. Points discussed here include the nature of RESOLVE "components"; principles for design, specification, and implementation of high-quality reusable components; the role of formal methods; modular reasoning about functionality and performance of component-based software systems; and several others.

The issues raised in Part I serve to highlight some fundamental differences between the RESOLVE approach to software component engineering and those taken by others.[1] To capture the distinctions and to support a more straightforward application of our principles, we designed the RESOLVE language. Some people claim that you can write good software in any language, but when it comes to reusable components this conclusion seems more a result of wishful thinking and an existence argument than of observation of practice. Engineering history shows that notation has a profound influence on reasoning and judgment regarding design. Parts II and III present our proposal for improvements on extant notation in computing, in the form of the RESOLVE language:

---

[1] Probably the best-known contributors to the popular practice of reusable software components are Grady Booch and Bertrand Meyer, whose recommendations, designs, and results are summarized, respectively, in Booch, G., *Software Components With Ada*, Benjamin-Cummings, 1987; and Meyer, B., *Reusable Software: The Base Object-Oriented Component Libraries,* Prentice-Hall International (UK) Ltd., 1994. There are other libraries of components in many languages and some interesting related work involving formal methods, e.g., specification and verification (which we discuss within the papers of the special feature). Booch and Meyer stand out, though, for their contributions both to the practice *and* to the theory of software component engineering, and for their visibility among software engineers who try to put software component reuse principles into action.

Booch's Ada components are widely recognized in the Ada community, and recently have been reworked for C++. The components include several hundred (mostly generic) Ada packages that implement variants of standard ADTs (stacks, trees, etc.) and some useful utilities. Informal explanations of most of these are given in Booch's book. The variants on basic functionality arise from Booch's taxonomy of general behavioral and implementation properties: whether a component is bounded or unbounded, whether it is concurrency-hardened, how dynamically-allocated storage is managed, etc. There are, for example, over 25 stack packages that differ along such dimensions. So the number of basic abstract components is much smaller than the total number of packages.

Meyer's components are written in Eiffel, Meyer's own object-oriented language. They include ones that are similar in basic functionality to Booch's ADTs, along with some interesting utility components for language processing, etc. Meyer writes (partial) "specifications" for some aspects of his components, but no abstract mathematical models are involved since expressions used as pre- and postconditions and invariants are meant to be run-time checkable. He uses Eiffel's inheritance mechanism both in a biology-like taxonomy to organize components (classes) and to separate interfaces from implementations. One result of this is a large number of operations per component. For example, there are over 20 things that you can do to any stack.

At the level of the prefaces of the above-mentioned books, it seems as though we might have written practically the same words as Booch or Meyer. But a deeper analysis reveals how profoundly different the RESOLVE approach is. For instance, Booch and Meyer do not write complete abstract specifications for component behavior (we do); they are not especially concerned with modular reasoning about, or verification of, component behavior (we are); sometimes their variables denote abstract values of objects and sometimes they denote references to objects (our variables always denote abstract values); and copying for data movement is the basis of all their designs (we use swapping). These central differences, and many others, lead inevitably — as we have explained in detail [Hollingsworth 92b] — to totally different component designs. To appreciate the extent of the differences, you need to examine the interfaces of some components. Comparing how each of us handles a Stack ADT [Part II] is a surprisingly good first step, but appreciating the full scope of the resulting improvements involves examining other reusable software components as well [Harms 91, Weide 91, Hollingsworth 92b, Sitaraman 93, Sitaraman 94b, Weide 94b, Weide 94c].

mechanisms and notations for writing abstract component specifications and concrete component implementations. We give several concise examples and provide pointers to other published work for more complex ones.

The RESOLVE language is primarily a research vehicle that has allowed us to understand better how to synthesize and formally express a number of important ideas about component-based software. But the RESOLVE work is not confined to an idealized research world. It has been applied to building components and applications in "real" languages such as Ada and C++, and in classrooms of students ranging from freshman university students through experienced practicing software engineers. In Part IV we show the Ada and C++ counterparts of some of the example components in Parts II and III. We have a library of dozens of such Ada and C++ components, some of which are available via the Internet.

An annotated bibliography of much of the RESOLVE work is presented in Part V. Most entries are papers available in the general CS literature, or technical reports (pre-prints or dissertations) available on-line through the World-Wide Web.

Our overall goals here are to introduce the RESOLVE work to those who are unfamiliar with it; to summarize the bigger picture for those who have seen bits and pieces of it; and to document the major features of the RESOLVE language. We generally omit detailed comparisons to work by other researchers, as this material appears in the cited publications listed in the annotated bibliography. Where making connections to significant related work is essential to setting the context or explaining a major point, though, we do include discussion and references in a footnote as on the previous page. (Fortunately, most such footnotes are less obtrusive than that one.)

2. Acknowledgments

The work described in the special feature has profited from contributions of many other members of our research groups and students in our classes, past and present. We regret that there are too many to mention everyone individually and remain within our page budget. We also wish to thank Will Tracz for the opportunity to develop this special feature, and Dieter Fensel, David Guaspari, Doug Kerr, Steve Wartik, Ben Whittle, and others (who remain anonymous) for their helpful comments on a draft version.

We also gratefully acknowledge financial support for our research from the following sponsors:

- *The National Science Foundation*: The work at Ohio State is supported under grant CCR-9311702; at West Virginia it is supported under grant CCR-9204461.
- *The Advanced Research Projects Agency of the Department of Defense*: The work at Ohio State is supported under ARPA contract number F30602-93-C-0243, monitored by the USAF Materiel Command, Rome Laboratories, ARPA order

number A714; at West Virginia it is supported under ARPA contract number DAAH04-94-G-0002, monitored by the U.S. Army Research Office.

- *The National Aeronautics and Space Administration*: The work at West Virginia is supported under grant 7629/229/0824.

3. The Authors

The authors of the papers in the special feature are listed below. We note only authorship (not affiliation, phone number, and Internet address) on the individual papers.

Paolo Bucci[2]
Stephen H. Edwards[2]
Wayne D. Heym[2]
Joseph E. Hollingsworth[3]
Joan Krone[4]
Timothy J. Long[2]
William F. Ogden[2]
Murali Sitaraman[5] (coordinator for the special issue)
Sethu Sreerama[5]
Bruce W. Weide[2]
Sergey Zhupanov[2]
Stuart H. Zweben[2]

---

[2] Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210; 614-292-5813; {bucci, edwards, heym, long, ogden, weide, sergey, zweben}@cis.ohio-state.edu.

[3] Department of Computer Science, Indiana University Southeast, New Albany, IN 47150; 812-941-2425; jholly@ius.indiana.edu.

[4] Department of Mathematical Sciences, Denison University, Granville, OH 43023; 614-587-6484; krone@sunshine.mathsci.denison.edu.

[5] Department of Statistics and Computer Science, West Virginia University, Morgantown, WV 26506; 304-293-3607; {murali, sethu}@cs.wvu.edu.