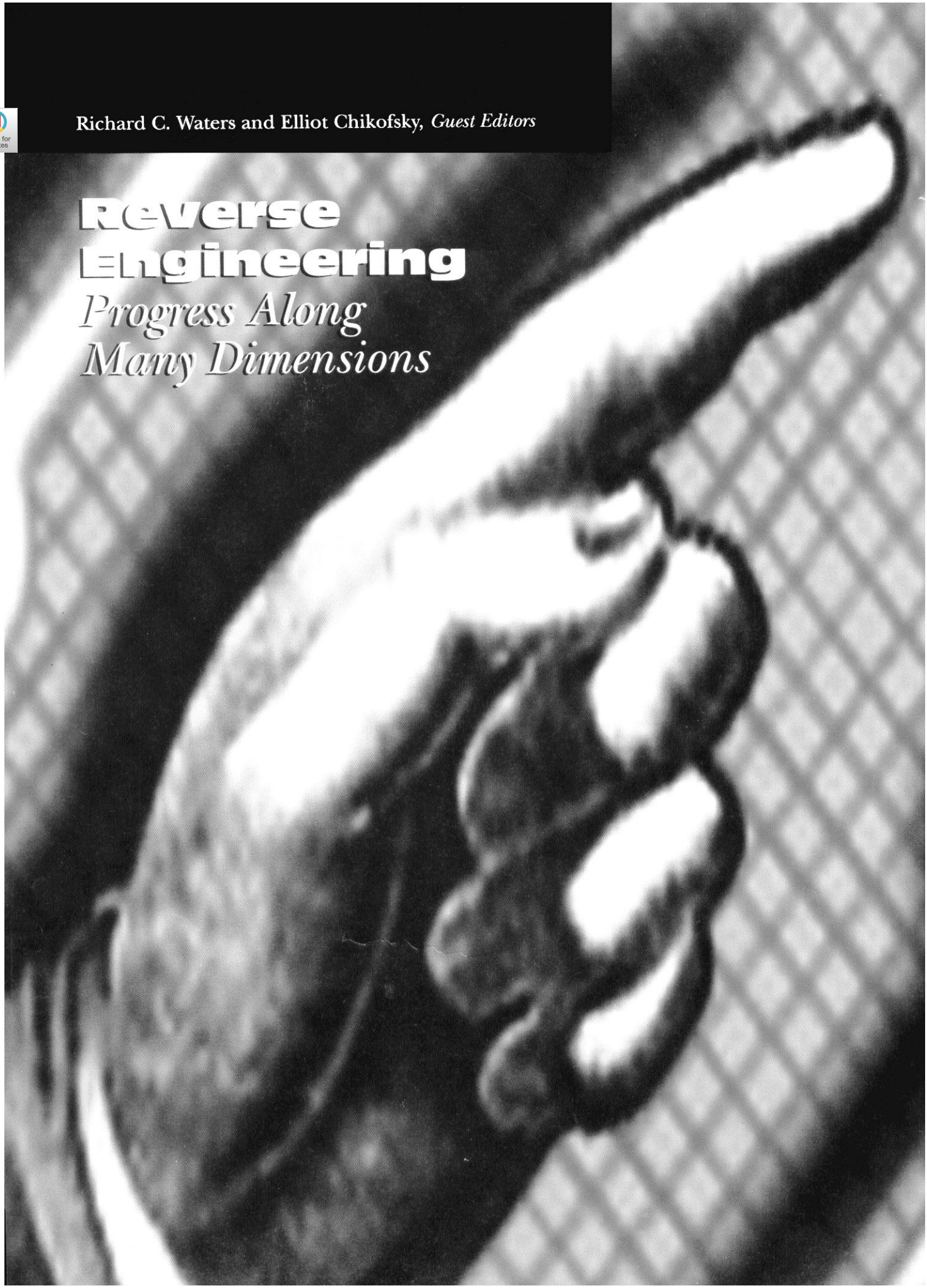
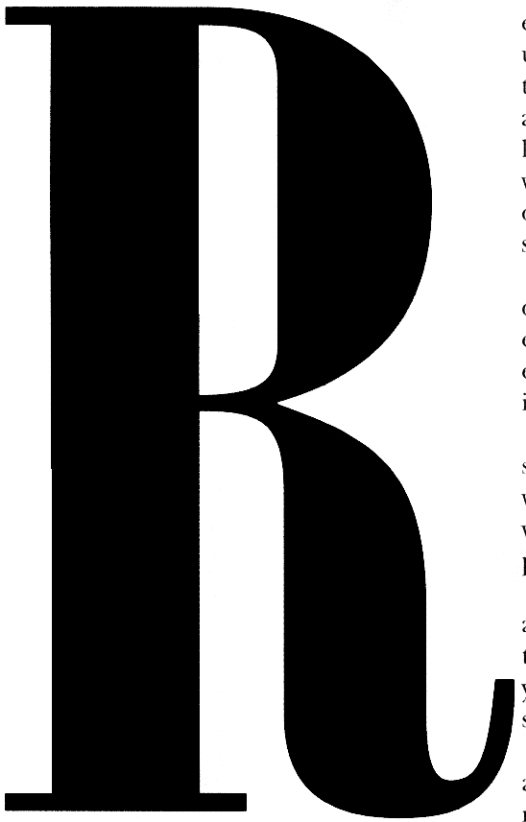
Richard C. Waters and Elliot Chikofsky, *Guest Editors*

Reverse Engineering

Progress Along Many Dimensions





Illustrations: Rico Lins Studio

Reverse engineering encompasses a wide array of tasks related to understanding and modifying software systems. Central to these tasks is identifying the components of an existing software system and the relationships among them. Also central is creating high-level descriptions of various aspects of existing systems. In line with this, the heart of research on reverse engineering is the development of tools and techniques for the analysis and representation of information about software systems.

The term "reverse engineering" is borrowed from hardware development, where it is typically applied to the process of discovering how other people's systems work. However, in software engineering, the term is used to describe the process of discovering how your own systems work.

At first glance, the need to reverse engineer one's own systems seems to be an admission of failure, If our systems did what we wanted them to, we would not need to change them, and so we would not need reverse engineering if we had sufficiently complete records of how our systems work.

Perhaps because of this negative aura, reverse engineering was a neglected part of software engineering for a long time. Under the banner of "get it right the first time," the focus for many years has been on software productivity tools to aid in the construction of new software systems.

However, there is a fatal flaw in the build-it-right-the-first-time approach. While it is always good to try and do a good job, it is not possible to get a system permanently correct, because it is not possible to predict now what you will want the system to do five, or even two, years from now. This is particularly true in a climate in which the cost of computers is dropping so fast that every year opens new horizons for what a system will be able to do. Further, while it is certainly valuable to capture information about how a system works for future use, it is not possible to predict now every question you are going to ask about the system in the future.

In any event, no matter what we might wish to be the case, the undeniable reality of software system development is that year after year the lion's share of effort goes into modifying and extending preexisting systems, about which we usually know little. That is to say, while many of us may dream that the central business of software engineering is creating clearly understood new systems, the central business is really upgrading poorly understood old systems. By implication, reverse engineering is arguably one of the most important parts of software engineering, rather than being a peripheral concern.

Over the past few years a general recognition of the importance of reverse engineering seems to have emerged, with work in the field growing by leaps and bounds. One outgrowth of this burgeoning interest was the convening in May 1993 of the first research conference devoted solely to reverse engineering. Under the sponsorship of the ACM and the IEEE Computer Society, this Working Conference on Reverse Engineering (WCRE), brought together academic and industrial researchers from around the world. The proceedings, available from IEEE CS Press, is the most comprehensive collection of work on reverse engineering to date. This issue of *Communications* contains revised versions of several of the best articles from the conference.

In This Issue

Reverse engineering attacks a range of problems from recovering the high-level architecture of a system, to understanding the details of the algorithms used, to ferreting out the business rules embodied in a program. It is applied for a variety of reasons from obtaining the information needed for rational system modification, to detecting the defects, to recovering components for potential later reuse.

The articles chosen for this issue were selected to show the breadth and potential of the field. Each one focuses on a particular aspect of reverse engineering, showing what can be done and how it can be valuable. Although no one article surveys the field as a whole, taken together, the authors in this special section give a broad feel for the kinds of work being done.

Any particular application of reverse engineering occurs within an organizational context and is needs-driven. Where a traditional collection of research papers would place articles on applications at the end, if they appeared at all, in this issue we take the opposite approach. Aiken, Muntz, and Richards examine the complexity of legacy systems within the U.S. Department of Defense and lessons learned from the reengineering of data requirements.

The central activity of reverse engineering is recovering information of a wide variety of types. Premerlani and Blaha explore the analysis of existing relational database structures to identify the underlying data model in an object notation. Ning, Engberts, and Kozaczynski describe a technique for identifying and extracting functional pieces of a Cobol program. For the general reader, the key importance of these articles is their presentation of various kinds of tools that can be used to extract information from programs.

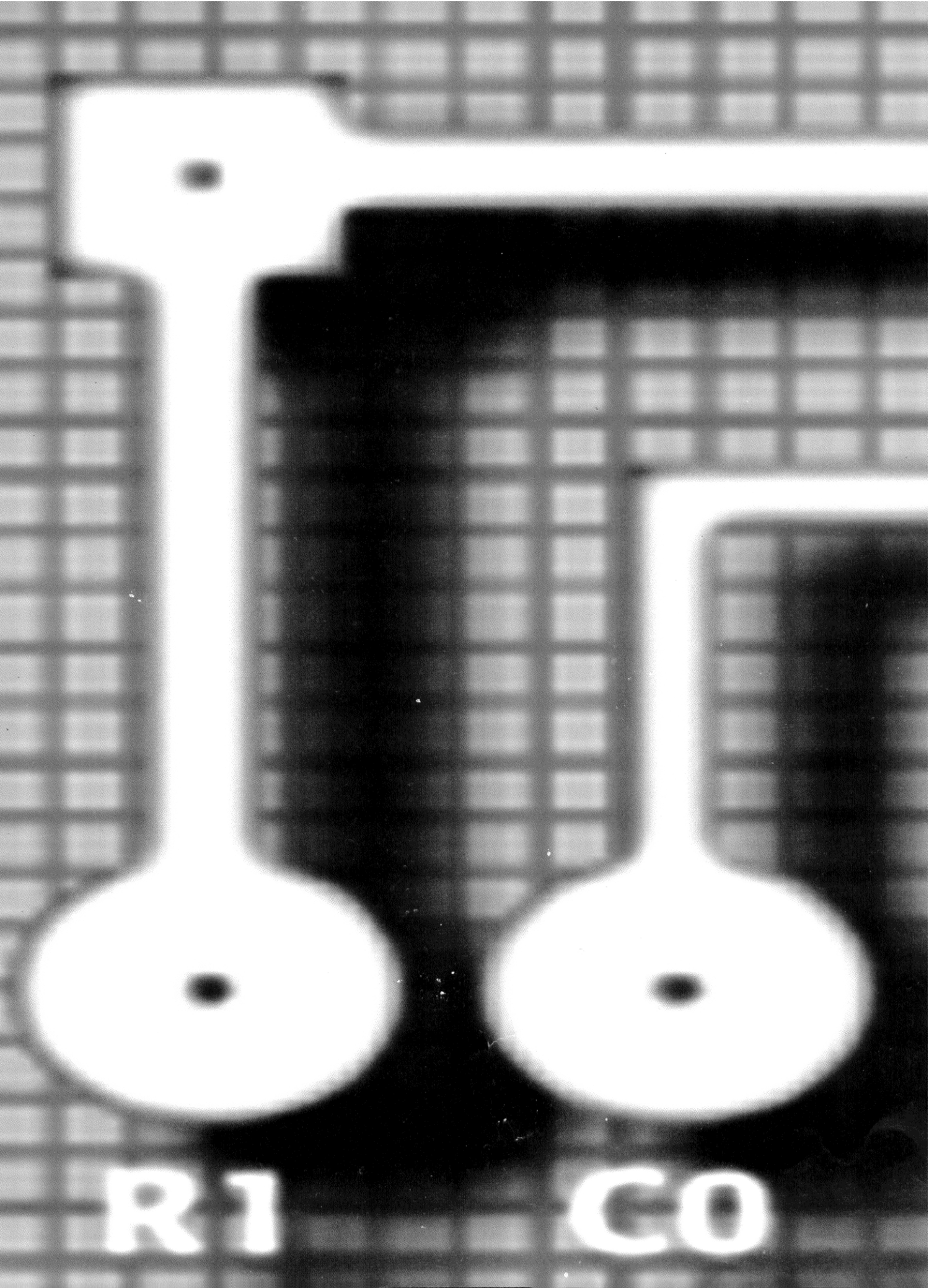
A theme that runs through much reverse engineering work is program improvement. Markosian, Newcomb, Brand, Burson, and Kitzmiller describe the generation of smaller, functionally equivalent modules from large Cobol programs. Their work demonstrates applying a tailorable commercial product to a particular reengineering strategy.

Most approaches to reverse engineering are semi-automatic in nature. Tools extract information, but people have to guide the tools and decide what information to look for. Biggerstaff, Mitbander, and Webster discuss some of the problems of relating implementation artifacts to the conceptual model of the human observer, and how these issues influence the direction tools need to take.

Under the rubric of program understanding, artificial intelligence researchers seek to create fully automated reverse engineering systems. Quilici examines an approach to recognition of detailed programming plans (patterns) that combines top-down and bottom-up strategies.

Richard C. Waters *is a senior research scientist at Mitsubishi Electric Research Laboratories in Cambridge, Mass. He may be reached at (617) 621-7508; email: dick@merl.com*

Elliot J. Chikofsky *is a principal at the DMR group in Waltham, Mass. and a lecturer in industrial engineering and information systems at Northeastern University in Boston. He may be reached at (617) 272-0046; email: chikofsky@comouter.org*


R1
C0