SOFTWARE COMPONENT ANALYSIS
As a long-time proponent of disciplined coding practices for safety-critical systems, I read Gerard J. Holzmann’s “The Power of 10” (Software Technologies, June 2006, pp. 95-97) with keen interest. Most of his rules are golden indeed, and our own rules similar to his have served us well in our projects. However, Holzmann’s Rule 7—Each calling function must check the return value of non-void functions, and each called function must check the validity of all parameters provided by the caller—baffles me a bit.

This practice could result in numerous, avoidable checks leading to poor runtime performance and code that will never execute. Dead code is often unacceptable for certification under standards like DO-178B.

The practice that makes better sense is to perform such checks only on external inputs and design functions to produce demonstrably correct output when inputs obey their constraints. Powerful static analysis tools are emerging in the marketplace—for example, CodeSonar for C/C++, Inspector for Java, Examiner for Ada—that developers can use to establish constraint adherence without resorting to runtime checks in most instances.

Perhaps the explicit checks practice should be reserved for reusable library functions where all possible uses are not known a priori and for those rare situations in which a formal proof is beyond the reach of today’s static analysis tools.

Vdot Santhanam
vdot@cox.net

The author responds:
The writer makes a valid point. Curiously, I noticed that many responses to the proposed 10 rules identify a different rule as qualifying for an exception. Exceptions to any one of the rules can indeed validly be claimed.

It is like the stop sign at an intersection of two deserted country roads. Do you stop, even though you can see for miles each way that no other car is coming, or do you drive through? Clearly, if you drive through, you can be ticketed, irrespective of whether it was safe to do so or not. The rule for stop signs could be reformulated to allow for exceptions—but it would lose much of its effect as a rule.

If it is easier to comply with a rule, rather than having to present detailed arguments why you shouldn’t, the rule will in most cases be followed—which is the effect we would like to achieve.

Rule 7 is meant to support the principles of locality and least surprise. This means that we want to be able to look at each function in a program as a logical unit and determine whether it is safe to execute or not. If, for example, the function de-references one of its arguments, there should be a protection in place against callers that pass a NULL pointer—whether or not this actually can happen in the current version of the code.

We want to avoid long chains of brittle reasoning that depend critically on one particular version of the code, under assumptions that may be long forgotten by the time the code is updated years later.

Gerard J. Holzmann
gholzmann@acm.org

DATA DICTIONARIES AND DEFINITIONS
I don’t understand why Neville Holmes is surprised by the data doughnut problem (“The Data Doughnut and the Software Hole,” The Profession, June 2006, pp. 100, 98-99). I would remind him that the computer science curriculum sponsored by the Computer Society basically ignores the entire issue of data analysis, data modeling, and database design.

Failed projects don’t list the incompleteness of the data definitions and database design as a reason for their failure. The failure of the data dictionary and the data definitions are essentially the reason for using the project metamanagement label. No one in the project ever realized that the term “net sales” had a different definition for each of the project stakeholders.

As Holmes says, the problem of developing the input and output software is only a data conversion issue, but that assumes that the enterprise-wide database has been well-defined, the processing assumptions are well-known, and the data dictionary is complete and readily available.

Creating a good data dictionary takes a mathematician’s analysis skills and a philosopher’s categorization ability. The time and cost to create the data dictionary do not produce any visible return on investment. Only when the dictionary is missing and the project fails do we recognize that a cost could have been avoided.

Rainer Schoenrank
rschoenrank@computer.org
Neville Holmes responds:

The point about what is taught in computer science is well taken. I suggest that the people who put together such curricula should be pressed to include more about data study.

However, I would claim, based on my own early experience (see “The Usefulness of Hindsight,” Computer, Nov. 2004, pp. 120, 118-119), that getting practical data dictionaries implemented is basically an upper-management responsibility, hence my “project metamanagement” classification, however much the details depend on the computing professional.

PRECISE COMMUNICATION

As a 30-year veteran of the software business, I agree fully with the points Neville Holmes makes in “The Data Doughnut and the Software Hole.” It’s gratifying to me to see that some of our “old school” ideas are being given due respect and that the weaknesses in today’s software methodology fads are being exposed.

I especially appreciate Holmes’s comments with respect to the imprecise use of language by software professionals today. It’s my belief that words such as “segregated” are not used because of their negative political connotations, even when used in a technical context. Further, I think that by using imprecise or inappropriate language, we actually hinder our own clear thinking.

It behooves all technical professionals not only to think clearly, but also to communicate clearly, if only because of the feedback loop to our own brains. If we are not thinking or communicating precisely as individuals, the communication among a project team almost certainly will contribute to failures, as Holmes describes.

My high school English teachers and my salesman father were correct to say that most of our problems today are due to poor communication. The technical problems are not nearly as difficult as the “people” problems.

Gary Rector
gary.rector@sap.com

We welcome your letters. Send them to computer@computer.org.