



How software
affects the world.
How the world
affects software.

Galen Gruman, Editor

HOW TO ASSURE QUALITY: DEBATE SHOWS DIVISIONS

ACHIEVING SOFTWARE QUALITY IS IMPORTANT, agreed debaters at the National Debate on Achieving Software Quality, but they disagreed on fundamental issues on how to do so. The eight debates, sponsored by the Society for Software Quality, were held Jan. 30-Feb. 1 in San Diego and drew about 100 people, mainly from the defense industry.

SHOULD SQA BE INDEPENDENT? One of the more controversial propositions was that independent software quality-assurance organizations are unnecessary. Opponents of independent SQA argued that this approach wastes resources, drives up costs, and fosters a counterproductive "us versus them" attitude. "The more people you toss at a problem, the less responsibility each takes," argued Nick Stewart, a software engineer at Rockwell International. Worse, he said, independence means "you are distant from the process forever, so the problems are going to be repeated in the next project."

An experiment at Lockheed where SQA was made part of the development team has shown success, said Suzanne Garcia, a software-quality engineer there who is part of the experiment. "I have developers asking to learn about quality assurance — that never happened when I was in the SQA organization," she said. Integrating SQA with development forces developers to "take back responsibility for the product," Garcia said, and requires management "to be responsible for the process."

Stewart and Garcia advocated the philosophy of total quality management, where each member of the team is responsible for assuring quality.

Other ills attributed to independence are that these organizations tend to be staffed with less experienced people and that senior management usually backs the development organization over the objections of the SQA staff, Garcia said.

"Independence lets you see the forest for the trees," answered Marilyn Bush, a systems and software engineer at the US National Aeronautics and Space Administration's Jet Propulsion Laboratory. "You need a group not colored by the heat of battle. You need an independent group to shake that complacency," she said. Bush described the success of independent SQA in NASA's space-shuttle software effort, which had an unusually low rate of 0.11 errors per thousand lines of noncomment source code.

She disagreed that independence necessarily re-

sulted in an adversarial relationship with the development team. The key is "to get in before the project begins and work with managers and developers throughout the life of the project," Bush said. "We're looking at a team approach," echoed Taz Daughtrey, a software-quality specialist in Babcock & Wilcox's nuclear-power division. He compared independent SQA to a sports team's coaching staff: "There is no coach in charge of winning. SQA is a team player with special responsibility for applying quality principles." Bush also argued that independent SQA and total quality management are complementary, not exclusive.

Acting as part of the team establishes the SQA staff's trustworthiness, while being independent gives the company recourse when a quality problem cannot be resolved between the development and SQA staffs, Bush said. "If SQA's role is second-guessing, it is not being managed properly," Daughtrey said, "and in the extreme cases, the ability to blow the whistle is valuable." If management ignores the warning, there is at least a record of objection for later historical or legal use, he said.

As to the cost of independent SQA, Bush said that NASA and US Defense Dept. statistics showed that it costs \$100 to find and fix an operational error early in development but \$10,000 or more late in the life cycle, so finding only eight defects early would pay for the salary and overhead for one SQA staff member. NASA estimates that establishing independent SQA saved JPL \$7.5 million, she said.

DOES USING ADA HELP? The Ada language and its development environment give developers of large systems the tools needed to productively "develop software and maintain it at a specified quality level," said Mark Dowson, a software scientist at Software Design & Analysis. Ada offers four main benefits:

- ◆ Clarity. Ada code is almost self-documenting, and its constructs support clarity. While a disciplined use of the language is "essential," there are tools to help enforce this.

- ◆ Quality. Ada removes programmers' freedom to make certain kinds of common errors, and its use of strong typing, data abstraction, exception handling, modularity, and concurrency all help developers make robust software.

- ◆ Reuse. Ada's support of reuse is "a powerful

Continued on page 103

Editor: Galen Gruman
IEEE Software
10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Internet g.gruman@compmail.com

QUALITY-ASSURANCE DEBATE

Continued from page 99

way to reduce costs.”

◆ Separation of concerns. Ada lets developers partition the design and coding efforts so team members can concentrate on their sections and compile them separately.

“If something sounds too good to be true, it probably is,” countered Randy Jensen, chief scientist of the software technology lab at Hughes Aircraft Ground Systems Group. “There is little proof that Ada has improved productivity,” he said. He cited a March 1989 study by the US General Accounting Office that showed that the Defense Dept. had not kept sufficient records to show Ada’s effects on productivity. The GAO tried to find 100 Ada-based projects for its report but found only 13. Of those, only three had been completed, and two of those were study products.

Jensen dismissed any claims of improved productivity based on counting lines of code because Ada has a 5:1 ratio of comments to code compared to 2:1 for Fortran, according to a NASA study. That NASA study, Jensen said, showed programmers using Fortran to be more productive than those using Ada when counting the number of noncomment statements produced rather than noncomment lines, Jensen said. Ada’s lower productivity continued even after developers had done several projects in the language, he said.

Jensen said the advantages Dowson cited were not Ada-specific. “C and C++ are used most in industry, where productivity counts. They support reuse and object orientation. Their development facilities compare to or surpass Ada’s. The existing tool sets for C, Pascal, and even Cobol surpass Ada’s,” he said.

The structured techniques and development methods in Ada are independent of

Ada, Jensen said, and can be used in other languages. The real key to improved productivity, he said, is to get developers to use such techniques whatever language they use. Less than 15 percent of the software industry uses structured programming, which has been around since the mid-1960s, he said, and less than 3 percent uses structured design, structured analysis, or object orientation.

Ada’s complexity is a reason to avoid using it, argued Dave Thomas, president of Object Technology and a computer-science professor at Carleton University. “There are 20 ways to write the same thing in Ada,” he said, “There’s a huge amount of syntactic baggage even for simple applications.”

Thomas also criticized the strengths Dowson cited. Because Ada is object-based rather than object-oriented, it straddles the line between procedural and object-oriented language, requiring mechanical translation, he said. “Strong typing is harmful: It provides a false sense of security [because] first-year students make type errors, not professional programmers,” Thomas said. And strong typing reduces reuse “because you need to instantiate each type used,” which also inhibits incremental evolution, he said.

The problems Jensen and Thomas cited were real — of Ada of the past, countered Glenn Hughes II, manager of government program operations at Rational and former official at the Defense Dept.’s Ada Joint Program Office. He cited the work of the Swedish firm Bofors Electronics in developing an integrated command and weapons-control system for ships. That work was completed after the GAO study was done.

Bofors designed reuse in from the beginning — “note the use of the word ‘design’” — for its 1.5-million-line system and then adapted this base system for six other types of ships in subsequent contracts, Hughes said.

Bofors estimated that it reused six times as much code in the Ada-based ship system than it had in previous systems written in RPG 2 and that system integration took half the time.

"Coding in Ada does not make a difference," Hughes said, "Ada facilitates programming with modern software-development techniques, which leads to significant productivity improvements."

WHAT IS TESTING'S ROLE? In defending the proposition that testing is the best way to detect and remove defects, Bill Howden, a computer-science professor at the University of California at San Diego, argued that "testing is not just looking at I/O." Instead, testing "compares expected versus actual behavior — via run-time execution, prototyping, tracing, etc. — during the whole process: specification, design, and coding."

Michael Fagan, a consultant who developed inspection techniques when at IBM, countered that "it is not feasible to test software completely. But we can — we do — inspect requirements, design, and code completely. Test cases are like throwing darts. The

question is the practicality, not the possibility, of complete testing." Also, testing "costs more and lets through more defects" than inspections, he said, because "testing finds faults, not defects." Once the faults are found, the cause must be identified, fixed, integrated, and retested. "Because inspections identify defects directly, repair is simpler and costs less," Fagan said.

"You can test exhaustively for some applications, like finite-state machines," rebutted Richard DeMillo, director of the Software Engineering Research Center at Purdue University. For other applications, "there are many methods for shrinking the space of faults, of likely inputs to something manageable."

"Testing will rarely find potential deadlocks in concurrent systems, failure to meet runtime deadlines, and failure to terminate," argued John Knight, a computer-science professor at the University of Virginia. Most important, he said, is "how do you know what tests to make and whether the output is correct?"

All four debaters agreed that testing has a

valuable role. "Testing is an essential phase in development as part of an overall approach to verification. It is a verification technique, not a debugging technique," Knight said.

IS REUSE PRACTICAL? Although reuse — whether of code, design, specifications, or tests — is an admirable goal, "in the real world, several things stop us from doing reuse," said Gary Sheinfeld, head of interoperability for the AWACS radar-system aircraft program at the US Air Force's Electronics Systems Division. Those roadblocks include liability, ownership, procurement regulations, the "not invented here" attitude, and not knowing what is available for reuse.

"The cost of reusing may be more than developing from scratch," added Jay Crawford, head of the embedded-technology office at the US Naval Weapons Center. "Even a 30 to 40 percent improvement in cost from reuse is not enough," he said.

Although the proponents of reuse offered no data to support their thesis that reuse lowers development costs, they argued that reuse was good practice.

"Economics dictate that a software contractor is going to try to avoid building all software from scratch," said Gordon Anderson, a lead engineer for the Tower Operator Tracking System at Logicon.

IS PROTOTYPING GOOD PRACTICE? "The reality is that you iterate. All engineering disciplines work this way," said Edward Miller, technical director of Software Research, in arguing that software needs to evolve as a series of prototypes. "Building it right the first time from first principles is a fantasy," he said.

"Prototyping is often used as an excuse for poor practice," countered Sam Redwine, assistant to the president at the Software Productivity Consortium. Other techniques — including market research, requirements analysis, performance engineering, management by customer as leader or by strategic alliance, and human engineering — are more effective, he said. Consultant Fagan was more blunt in his attack on prototyping, calling it "the dreaded H word: hacking."

A prototype is cobbled together and enhanced over time to add the functions that should have been there in the first place, Fagan said.

"A prototype is a solution looking for a problem. It's cobbled together and enhanced over time to add the functions that should have been there in the first place," Fagan said.

Miller countered the hacking charge by advocating that prototyping be based on sound engineering practice, such as requirements analysis. Fagan and Redwine conceded that prototyping is appropriate if engineered for concept validation for competitive bids.

But "people aren't smart enough to develop complex systems without evolutionary development," said Bill Rotzheim, a senior associate at Booz, Allen & Hamilton.

Another reason to prototype is the "reality of funding," Rotzheim said, "You need to show something every year or quarter that is useful. Also, requirements change. Functions should be delivered incrementally until the customer says he has all he wants."

OTHER AREAS. The other three debates were less controversial:

◆ In debating whether rigor is the most important aspect of practicing any methodology, debaters agreed that choosing the most appropriate method was the most critical factor. But the debaters disagreed on how rigorously the chosen methodology should be followed, apparently because of disagreement on what "rigor" means. Opponents claimed it meant slavish execution, proponents said it meant disciplined practice.

◆ Debaters arguing over whether Defense Dept. software unnecessarily required too much documentation agreed that the level required was higher but was not unnecessary. Ben Scheff of Raytheon argued that the real problem was that engineers were doing documentation, rather than documentation specialists. Allan Ott of Logicon argued that developers frequently overdocumented their software, apparently unaware that DoD-Std-2167A encourages contractors to tailor the documentation for each project. He acknowledged that contractors may fear suspicion from the contracting agency that the tailoring is an attempt to avoid requirements.

◆ The four panelists debating whether software under development should be configuration-controlled immediately upon successful completion of unit test agreed that configuration control should be part of the entire development effort and not be introduced only at unit test.