Director’s Notes

“A cost-effective prevention program first requires accurate error detection and analysis to understand where, how, and why defects are inserted.”

This quote comes from our lead article, “Software Quality and Testing: What DoD Can Learn from Commercial Practices,” written by LTC Mark R. Kindl of the US Army Research Laboratory’s Software Technology Branch. We at the DACS thank LTC Kindl for submitting his article. We encourage each of you to submit software engineering-related articles to the DACS for publication consideration in future issues of the DACS.

Several upcoming conferences are listed on page 6 of the newsletter. Also, located on page 4, is an invitation to participate in the work of the IEEE Software Engineering Standards Study Group. Designed to formulate strategic and tactical objectives, a list of study group areas is included in the invitation. We encourage your long-term involvement in this endeavor.

In our regular feature section, CLOSE-UP CORNER, Mark Costello, a member of the DACS staff, defines, describes and graphically and mathematically depicts Artificial Neural Networks in part one of this two part series. Should you be interested in obtaining greater details on Artificial Neural Networks, contact us for information on obtaining a DACS Technical Report on the subject prepared within the past year.

We also direct your attention to the announcement of the upcoming 8th Knowledge-Based Software Engineering (KBSE 8) Conference, slated for 20 - 23 September 1993 in Chicago. It’s never too early to plan for your attendance.

Questions, comments, or technical queries? Contact the DACS staff at (315) 734 - 3696 for a prompt reply.


by LTC Mark R. Kindl, Army Research Laboratory, Software Technology Branch

Historically, software testing was the process of exercising a computer program to verify that it performed as required and expected. The strategic goal of software testing was to demonstrate correctness and quality. It is now known that this view of testing is not correct. Testing cannot produce quality software, nor can it confirm correctness. Testing can only verify the presence, not the absence, of software defects. Yet, the difficulty of testing and the impracticality of correctness proofs have often driven us to the dangerous perception that if testing does not find defects, then the software is correct.

In the early 1980s, software testing concepts were neither well-developed nor well-understood [1, p. 39]. While testing techniques were many, supporting theories were few. Even worse, little or no guidance existed for making intelligent choices of technique(s) [2, vol. 1, p. 24]. The Department of Defense (DoD) and industry gathered much empirical evidence during the 1980s to justify many software quality and software development techniques. As a result, the scope of software testing has evolved into an integrated set of software quality activities that cover the entire life cycle [3]. Software tests now take different forms and apply to all software products including requirements, design, documentation, test plans, and code. Each test contributes to a total quality assurance plan. Quality assurance focuses on the front of the development process and emphasizes defect prevention over detection. A cost-effective prevention program first requires accurate error detection and analysis to understand where, how, and why defects are inserted. Though testing cannot prevent errors, it is the most important method for producing error data necessary to guide process improvement. The

(continued on page 2)
following extract from the 1992 Software Maintenance Technology Reference Guide [4], however, summarizes the difficulty of testing: "Software implementation is a cozy bonfire, warm, bright, a bustle of comforting concrete activity. But beyond the flames is an immense zone of darkness. Testing is the exploration of this darkness."

The conclusions presented in “Software Quality and Testing: What DoD Can Learn from Commercial Practices” report are not revolutionary, but they may be surprising. DoD knows how to produce quality software. There are a few contractors who produce quality software using many of the policies published in DoD standards. These documents describe the need to focus on quality activities early in the software life cycle. Developers and verifiers should identify and remove errors during requirements definition and design so that they do not enter the code, where finding and fixing defects is extremely expensive. For management information and command/control systems this is a particularly difficult task because most requirements for these systems are based upon human demands which are highly subjective, easily influenced, and therefore, very dynamic and difficult to state precisely.

Although not common practice yet for software development, quality control methods adapted from the factory paradigm [5] may have the greatest potential to move software production from an art to a true engineering discipline [6, 7, 8, 9]. Both the products and the development process should be subjected to these procedures. To engineer quality into the software products requires that we inspect/test and remove defects from requirements, design, documentation, code, test plans, and tests. Quality control of the development process requires that we establish standard procedures to measure defects, determine their root causes, and take action to prevent future insertion. Such a process is self-correcting, and future measurements will provide convincing evidence of cost-effective improvement. In summary, software quality improvements are evolutionary and require that we control, coordinate, and feedback into three concurrent processes: the software development process, the error detection process (testing life cycles), and the quality improvement process. Figure 1 depicts the relationships between the processes in the software life cycle.

Figure 1 - Software Quality Control
A few corporate organizations have successfully implemented these procedures [10, 11, 12, 13, 14, 15, 16, 17]. The common key element in these successes is organization-wide commitment to a quality attitude and disciplined life cycle procedures. Within the DoD, however, the perception persists that such practices are not cost-effective. Simply mandating their use has not been adequate. Even if enforced, the techniques can be undermined, and neither software quality nor the perceptions will change [10]. DoD must jump-start these procedures with an active campaign to establish and nurture a quality attitude both internally and in its contractors.

In order to initiate the production of higher quality software within DoD, we recommend the following actions:

- Actively motivate a software quality attitude in DoD and government contractors through management commitment, incentives for process improvement and quality, and technical training. Make quality as visible as the software product, its cost, and its schedule. For every change to a software product, cost, or schedule, DoD project managers must give equal consideration to the corresponding cost of the effect on quality.

- Motivate and make standard use of formal inspections for all software products (requirements, documentation, design, code, test plans, tests).

- Users, developers, and verifiers should jointly analyze requirements to ensure they are clearly documented, implementable, and testable. The formal analysis of quality should be an integral part of this effort. A joint relationship should continue throughout the software life cycle. Eventually, this effort should result in documentation or data that directly cross-references test cases to requirements and code. At the same time, both developer and verifier should independently plan, design, develop, inspect, execute, and analyze the results of software tests.

- Measure and document errors throughout the life cycle. Establish a formal defect prevention program which empowers developers and verifiers to analyze the causes of error and enact improvements to their own local development processes that will prevent future error insertion and enhance detection processes.

- Evolve Computer-Assisted Software Engineering (CASE) tools to support all aspects of software development, testing, and maintenance. DoD should permit organizations to introduce standard CASE tools gradually in piece-meal fashion. An organization should purchase, train, and employ only those tools for which sub-processes are defined in writing. Start small and allow adequate time to learn and gain experience. Purchase and integrate a new tool only when users understand the manual procedure the tool will automate, and the benefit of automating it.

We can summarize our conclusions in regard to software testing in DoD in two fundamental ideas. First, DoD knows how to produce quality software at low cost. This is because organizations such as DoD Software Test and Evaluation Panel (STEP), Army STEP, and the Software Engineering Institute (SEI) have already researched and documented policies for DoD. Second, quality cannot be tested into software. Only a well-defined, well-disciplined process with a continuous improvement cycle can ensure software quality. Testing, however, cannot be underestimated. Systematic testing activities that detect errors earliest in the life cycle are necessary to drive process improvement and optimize the development of quality software. Such testing methods as formal inspection find defects early. This enables cost-effective error resolution, identification and removal of defect causes, and therefore, prevention of future defect insertion. If practiced with discipline, such methods can evolve a self-correcting software development process that is stable, modeled, measured, and therefore, predictable. This development process engineers quality software faster at reduced cost.

“Software Quality and Testing: What DoD Can Learn from Commercial Practices” discusses software testing practices, and more specifically, why and how one software developer's practices achieve high quality. Along the way, DoD policies, instructions, and guidance are related to the developer's practices. Current DoD initiatives are discussed which will impact software testing and quality. Specific recommendations are presented for software testing and quality within DoD. The author believes that these recommendations have the potential for immediate value to DoD.

To obtain a copy of "Software Quality and Testing: What DoD Can Learn from Commercial Practices," ASQB-GI-92-012, 31 August 1992, send a written request to:

LTC Mark R. Kindl
Army Research Laboratory
Software Technology Branch
ASQB-GCI
Georgia Institute of Technology
115 O’Keefe Building
Atlanta, GA 30332-0800
(404) 894-3110

References


   (1) Final Report and Recommendations
   (2) Software Test and Evaluation: State-of-the-Art Overview
   (3) Software Test and Evaluation: Current Defense Practices Overview
Software Engineering Standards Study Group Seeks Participants

The need to establish a study group to advance the future of software engineering standards was recognized in 1991 at the IEEE Computer Society Fourth Software Engineering Standards Application Workshop. According to Mr. Leonard L. Tripp, Software Engineering Standards Study Group (SESS) Vice-Chair, "the mission of the study group is to advance software engineering as a discipline through the implementation of effective standards which will be consistent and compatible in content and purpose." The SESS has identified the following two tasks necessary for accomplishing its mission:

1. Identify the long-range development and harmonization requirements for Software Engineering Standards (SES), and
2. Prepare and coordinate a strategic plan for meeting the requirements.

The SESS has met several times since its inception: May 1991, San Diego, CA; October 1991, Nashville, TN; February 1992, San Francisco, CA; March 1992, Huntsville, AL; Herndon, VA; and November 1992, Minneapolis, MN. Recent activities have focused on the development of the Software Engineering Standards Long-Range Plan. The purpose of the plan is to "(1) define in users' terms what expectations software engineering standards should satisfy, (2) allocate expectations to a framework based on type of standard, (3) derive requirements for standards from the expectations, and (4) define the requirements for software engineering standards to be developed during the 1990s in standards developer terms." The Software Engineering Standards Long-Range Plan will be sent out for ballot in March 1993.

The SESS is currently soliciting participants and reviewers for a series of SES study groups. These groups will formulate strategy and tactical objectives for the following areas: Software Testing; Software Reliability; Software Measurement; Standards Taxonomy; Application Programming Interfaces; Programming Languages and Software Engineering Standards; Software Life Cycles for Non-traditional Development Approaches; Conformance Testing and Software Engineering Standards; Criticality and Software Engineering Standards; Process Maturity and Software Engineering Standards; Integral Processes; and Improvement of Existing Software Engineering Standards.

All interested parties are invited to participate in SESS activities. Contact Mr. Leonard L. Tripp, SESS Executive Vice-Chair, at the following address for additional information on SESS activities:

Mr. Leonard L. Tripp
Boeing Commercial Airplane
PO Box 3707, MS 6Y-07
Seattle, WA 98124
(206) 477-3326; Fax (206) 477-3330
Internet: lltripp@kgv1.boeing.com
Neural Networks: What Are They and How Are They Used
by Mark G. Costello, Kaman Sciences Corporation

Neural Network: (1) a network composed of living nerve cells and other biological structures that make up the neural system of living beings, (2) a data processing system whose architecture is inspired by biological neural systems.

The neural network this article discusses are of the second variety and are more accurately referred to as artificial neural networks, Figure 1b. The study of these neural networks can be viewed from two aspects: to model biological systems and to model technological systems. In technological modeling the goal is to utilize biological neural concepts to inspire new and effective computational methodologies. This article will discuss technological modeling where ideas are borrowed from biological neural networks, but are not constrained by biological fidelity.

Figure 1 - (a) Natural Neural Network, (b) Artificial Neural Network

Work on neural networks has a long history. Development began back in the 1940s with the work of McCulloch, Pitts, and Hebb. Some work continued in the 1960s with Rosenblatt and Widrow among others. The work trailed off for a number of reasons, but recent developments in new net topologies and algorithms has led to a new resurgence.

Neural networks are not programmed with a computer language the way conventional computers are, but are literally trained to behave the way we want them to. How do these systems work if they are not programmed? What can they do well, and what do they not do well? What kinds of applications can we expect them to perform? To answer these questions, let us begin by describing the characteristics of a basic neural network.

A Neural Network

A neural network is composed of processing elements and connections. Connections between processing elements each have an independent coefficient called a weight. This weight amplifies, attenuates, and can possibly change the sign of the signal sent over the connection. In most networks the weights are adaptive.

The topology of the connections among the processing elements influences what each data element receives and as a result, the information processing functions the neural network can carry out. Some early researchers tried randomly interconnecting large numbers of elements, without much success. As we now know, even the brains of snails are structured devices. Designers have often found that an appropriate arrangement is that of interconnected layers of elements. Although there are uses for networks containing only one layer, or even one element, most applications require networks of two, three, or more layers. A common connection topology is to connect each element in layer n with every element in layer n+1. An example of such a “fully connected” three layer neural network is shown in Figure 2. Other topologies include partially connected layers; intralayer connections; feedback connections; and nonlayer connections.

Figure 2 - A three layer, fully connected, feed forward neural network.

In the example of Figure 2, the network accepts two inputs from the outside world. The first layer of processing elements are a degenerate case where they simply accept an input and distribute it to the nodes in the next layer. Each element in layer 2 accepts input from all layer 1 elements, attenuated by the corresponding weights for each element pair, and then generates a single output. Layer 2 is often referred to as the “hidden layer” or the “middle layer” in the literature. There is only one element in layer 3. It accepts the input from all layer 2 elements, attenuated by the corresponding weights, and then generates a single output signal for the entire neural network.

Each processing element has many input signals, but only one output signal as shown in Figure 3. The output signal can fan out along many connections to provide input signals to other processing elements. To compute its output, each processing element applies a mathematical formula, known as a transfer function, to its inputs. Each processing element is completely independent of other elements and as such is a candidate for parallel computation. There are many ways to process the inputs...
and many transfer functions. A common approach is to sum the weighted inputs and apply an S shaped (sigmoid) transfer function with asymptotic limits at 0.0 and +1.0, as shown in Figure 4.

Figure 3 - Processing Element

Neural networks are usually simulated on standard serial computers. Parallel CPUs and special hardware on which neural networks can be implemented more efficiently are available, for applications where performance is critical.

Figure 4 - Sigmoid Transfer Function

In the next issue, how neural networks are used will be described. All references will appear with the conclusion of the article in the next issue.

DACS Participation Plans: What Are They and How They Can Help You

The DACS Full Service Participation Plan provides full access to DACS resources through payment of a single participation fee. Participants can request consultations, custom bibliographic searches, and DACS publications. Participants receive one copy of each publication when issued, discount privileges, access to DACS resources, and account maintenance. A DACS Full Service Participation Plan can be opened depositing a minimum of $500 (the maximum to be determined by the requester). The DACS will send the user a summary of plan purchases on a quarterly basis. The Plan is open to all US Government agencies, government contractors, commercial users, laboratories, and educational institutions within the United States. Additionally, DACS products and services may be acquired by a simple phone call from the authorized member. This service allows users to obtain important information in the shortest amount of time.

Contact Judy Ross at (315) 734-3696 for more information on how you can take advantage of this service.
DACS Product Order Form

The DACS uses United Parcel Service (UPS) for shipment of some products. Please indicate a street address as packages cannot be delivered to a PO Box by UPS. Send completed order form and payment to the following address:

Data & Analysis Center for Software
PO Box 120
Utica, NY 13503
ATTN: Products/Services Order

<table>
<thead>
<tr>
<th>Name:</th>
<th>Title:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company:</td>
<td>Mail Code:</td>
</tr>
<tr>
<td>Address:</td>
<td></td>
</tr>
<tr>
<td>City:</td>
<td>State:</td>
</tr>
<tr>
<td>Zip Code:</td>
<td></td>
</tr>
<tr>
<td>Telephone Number:</td>
<td>FAX Number:</td>
</tr>
</tbody>
</table>

[ ] DACS Newsletter  [ ] CAMP Information Package
[ ] ACEC Information Package  [ ] DACS Information Package
[ ] ACS Information Package  [ ] User's Guide to Bibliographic Services
[ ] AFATL Ada Compiler Information Package  [ ] User’s Guide to DACS Products and Services
[ ] AFIS Information Package

* These items are free of charge.

<table>
<thead>
<tr>
<th>Product (Name/Format)</th>
<th>Unit Cost</th>
<th>Number Ordered</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[ ] Check payable to “Kaman Sciences Corporation” is enclosed $  
[ ] Apply this order to DACS Full Service Participation Plan #  
[ ] DD Form 1155, Vendor Identified as “Kaman Sciences Corporation, Data & Analysis Center for Software”

If you have any questions regarding our products and services, please contact:

Ms. Judy Ross
(315) 734-3696
8th Knowledge-Based Software Engineering (KBSE 8) Conference

The 8th Knowledge-Based Software Engineering Conference, being held at the Midland Hotel in Chicago, Illinois, on 20-23 September 1993, will begin with a day of tutorials. Following the tutorials, there will be two and one half days of paper and panel presentations as well as demonstrations.

The 1993 conference will provide a forum for researchers, practitioners, and developers to exchange ideas and results regarding the application of artificial intelligence and knowledge-based techniques to all aspects of software development. The conference scope will cover activities related to software, including planning, domain modeling, reuse, specification, design, coding, documentation, evaluation, prototyping, testing, maintenance, and re/reverse engineering.

For more information about this conference contact Barb Radzisz at the DACS or utilize the KBSE info-line. The KBSE info-line is an information service providing email access to various information about the conferences as well as Rome Laboratory’s Knowledge Based Software Assistant (KBSA) project. Information currently available includes a background and history of KBSA, a summary of past conferences, a KBSE reading list, and all information regarding the next active KBSE conference (KBSE8) including the Call for Papers, registration form, and program. This service can be accessed by sending mail to kbse-info@cs.rpi.edu. You will receive an automated reply which details how to use the service. Changes in the available information are broadcast to the KBSE mailing list.

DACS Newsletter

DACS Newsletter is the current awareness publication of the Data & Analysis Center for Software (DACS), a Department of Defense (DoD) Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC) under the DoD IAC Program. DACS is technically managed by Rome Laboratory (RL). The Contracting Officers Technical Representative (COTR) for the DACS is Mr. John Palaimo, Rome Laboratory, RL/C3C, Griffiss AFB, NY 13441-5700. He may be reached at (315) 330-4477.

Kaman Sciences Corporation manages and operates the DACS, serving as a centralized source for current, readily available data and information concerning software engineering and software technology.

Suggestions and submissions for DACS Newsletter articles are welcome. Please send your suggestions or submission to the following address. Inquiries about DACS capabilities, products, services, or comments regarding this publication may be addressed to:

Data & Analysis Center for Software
PO Box 120
Utica, NY 13503
(315) 734-3696, fax (315) 734-3699

Director, James J. Reed, (315) 734-3668
Editor, Barbara Radzisz, (315) 734-3667
Inquiries, Judy Ross, (315) 734-3696