BY JEFF HOLTZMAN

How to Become a Better Coder

MY APRIL COLUMN ("HOW NOT TO BE A BAD CODER") RESONATED WITH LOTS OF READERS, MANY OF WHOM HAD WAR STORIES OF THEIR OWN TO SHARE. SEVERAL PEOPLE, HOWEVER, ASKED RELATED QUESTIONS, WHICH IN ESSENCE BOILED down to this: "I know my coding is not as good as it could be. How can I make it better?"

As I began thinking about it, the question appeared more and more intriguing. Much of my initial thinking centered around the technical aspects of programming. (Learn CPU architecture, memory architecture, object-oriented programming, . . . etc., etc., etc., *ad nauseum*.)

Later I realized that all the technical things I had been considering were really limited aspects of the overall answer. I also realized that there is not a single one-size-fits-all answer.

If you assume that programming ability follows a normal distribution curve, we can simply lop off both ends. Those at the top don't need any help, and those at the bottom won't benefit no matter what we do. So our real audience is the 80% in the middle; those individuals exhibit a broad range of programming skills, a broad range of other skills, broad knowledge of other domains (both technical and practical), and a broad range of personality types. In most real-world situations, the other issues will be as important, if not more so, than raw technical ability.

That's not to say that a narrowly focused specialist with the personality of a dead moose can't succeed. However, well-rounded individuals will always be more valuable both to themselves and to the businesses that hire them.

Programming is Not Just Programming

The reason for that last statement is that in most real-world situations, programming is not just programming. It is not a "pure" activity, without constraint, like proving theorems. Rather, it is an engineering activity that almost always involves complex trade-offs among cost, schedule, product quality, design purity, resource usage, speed, integration, and other factors.

One key element of becoming a better programmer is to be able to recognize, evaluate, and act on each element of the trade-off matrix. That type of knowledge can only be gained through experience. But if you just rush through your programming experience like a freight train through the countryside, you won't learn anything. Rather, you will be wasting your time. To avoid wasting time, I suggest the following:

After each "increment" of programming experience, which could be as large as an entire project or as small as a single line of code, ask yourself what went right, what went wrong, and what you can do to maximize the former and minimize the latter. Then act on the answers that you come up with.

If you can retain a receptive state of

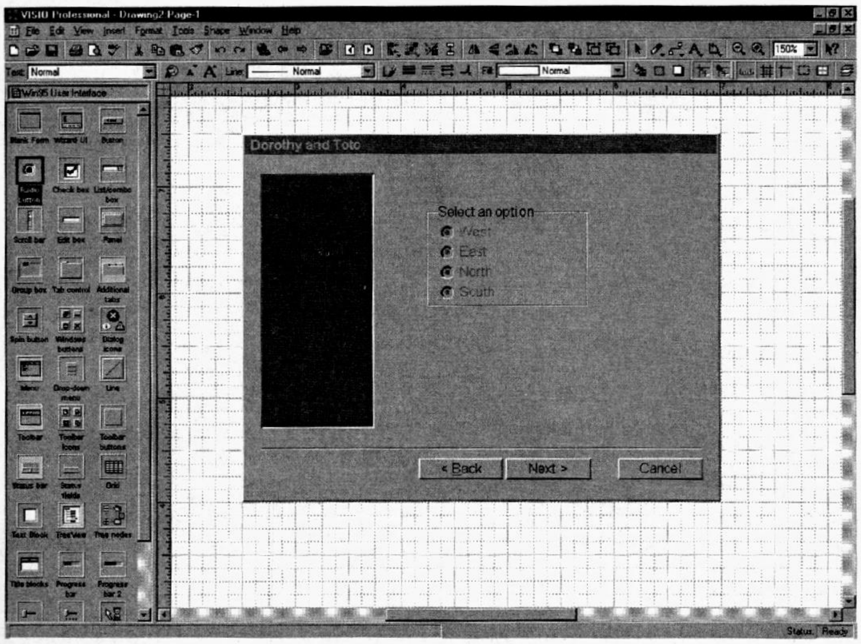


FIG. 1—IS THIS A NEW WIZARD FOR THE INHABITANTS OF OZ? No, it's a Visio user-interface design template. Visio is the drawing program for people who work with ideas.

mind, you might find that you develop the ability to "receive messages" in real-time about your work process. (Yes, you are both transmitter and receiver.)

Cost and budget pressures serve to lower the signal-to-noise ratio of those messages. Cost and budget also tend to increase the stress factor, which gives a positive feedback loop. In other words, as stress increases, receptivity decreases, which lowers performance, which increases stress, and so on.

You'll find that most of what you learn has nothing to with technical knowledge. The most valuable items concern the setting or context in which you do your work, including the people you work with and for, the types of problems you work on, time and financial constraints, and how you approach the problems you encounter.

Of course, without some modicum of technical ability, all that other stuff is irrelevant. If you're not a technical genius to begin with, you're not going to become one. But whatever your skill level, you can become more effective, more efficient, more valuable, more highly paid, and more satisfied.

Hints

To help you along in the process of becoming a better coder, here is a selection of thoughts focused on how to improve competence, technical and otherwise:

Don't reinvent the wheel. Probably 99% of all application-related computing problems have already been solved. Be aware of prior solutions, so you can either reuse or improve them. Being aware of prior solutions also helps you to not embarrass yourself with an incorrect or inefficient solution.

View problems as systems. No man is an island. Nothing exists in isolation. Everything can potentially affect everything else, particularly in software. Get in the habit of seeing systems as collections of interactive components whose collective behavior defines the nature of the system.

Read a lot. Stay on top of what the current issues, trends, and even fads are in your area of interest. Know what solutions already exist, and what their strengths are. If you have any interest in developing your own products, look for gaps or products with major limitations.

Write a lot. Practice doesn't make you perfect, but it keeps you mentally limber, and it puts you in the frame of mind to receive process-improvement messages.

Continually reinvestigate fundamentals. Cramming for the CS-101 final may get you through college, but it won't make you a better programmer. While I'll be the first to admit that it's hard, occasional reviews of algorithms, data structures, language fundamentals, and the like will improve your problem-solving ability.

Keep priorities straight. Everyone who has worked in a technical position in industry for longer than five minutes has gotten sidetracked into technical debate about (and even development of) subsidiary features that do not have a major impact on the overall product. Know what your goals are, even if your manager doesn't. Always keep those goals in mind, and you will succeed.

Only optimize where it's worth it. A corollary of the preceding is that not all parts of any given project are equally performance sensitive. Instrument your code to detect bottlenecks, and focus there. Any time you're optimizing without a specific objective in mind, you're squandering resources.

Technically interesting does not equal value to the project. Another corollary is that all components of a project must work, not just the "interesting" ones. Don't fool yourself about what needs to be done. Always keep the forest in mind while working on individual trees.

Strive for simplicity; be suspicious of complexity. Don't be simple-minded, but seek to recognize (in the problem domain) and create (in your code) patterns. That's really what object-oriented programming is all about: Creation of a hierarchy of patterns, with the hierarchical structure itself informing and enforcing relations among the elements of the hierarchy. If you can't explain something, it's not simple. Nor do you understand it well enough.

Stay in touch with the community. Know what people in your physical locale and mental domain are thinking and feeling. If you find yourself thinking and feeling vastly different things, you may want to change jobs.

Work with a mentor. Lose your ego. Get your code reviewed by experts. Learn from what they say, and don't take it personally. Try to institute code reviews and design walk-throughs on the job. Younger members will learn, and senior members (and management) can stop little problems from becoming big ones.

Practice "spiral learning." Learn everything you can about components. Get a broad background, but also focus in detail on several complex, specific examples. Then learn everything you can about a single small system. Then tackle a larger system. Then another. Then another. With maybe a dozen small systems under your belt, you'll be ready to attack system integration. (Here learn does not mean study. It means build and make work. If you fail occasionally, so be it; you'll still learn.) All that will keep you busy for ten to twenty years; then you'll be ready to lead and advise others.

Win95 Port I/O Revisited

Last month I presented a technique for directly accessing I/O ports under Windows 95. In the back of my mind was a rule that I probably should have made explicit, to wit: The proper way to access ports in Win95 is via a virtual device driver, or VxD. The subject of VxDs is complex. A good source for understanding VxD architecture is Andrew Schulman's *Unauthorized Windows 95*. For coding VxDs, you'll almost certainly want to get a copy of Vireo's VToolsD.

Visio 4.5

Visio is the drawing program for anyone who needs to represent ideas pictorially. I was enthusiastic about the program when it came out in 1992; I remain as enthusiastic as ever.

In a nutshell, Visio provides a stencil-based drag-and-drop drawing system. Stencils hold shapes, which range from circles and rectangles to transistors and

continued on page 60

RESOURCES

Unauthorized Windows 95
by Andrew Schulman ($39.99)
IDG Books, 1994
ISBN: 1-56884-305-4

VToolsD ($495)
Vireo Software, Inc.
21 Half Moon Hill
Acton, MA 01720
Tel: 508 264-9200
Internet: www.vireo.com.

Visio Technical 4.5 ($299)
Visio Professional 4.5 ($349)
Visio Corporation
520 Pike Street
Suite 1800
Seattle, WA 98101
Tel: 800 248-4746, 206 521-4500
Internet: www.visio.com.

device (CCD) has 350,000 square pixels. With its support for the square 1:1 pixel VGA standard used in PCs, the CCD eliminates the need for post-processing in the camera. A unique RGB color filtration method allows for accurate color reproduction.

The digital camera delivers full-frame, color images with 640 by 480 resolution. The DS-7 features a Fujinon lens and offers automatic exposure. Other features include portrait, landscape, macro, and low-light settings.

The DS-7 digital camera with LCD monitor has a suggested retail price of $699.

FUJI PHOTO FILM U.S.A., INC.
555 Taxter Road
Elmsford, NY 10523
Tel: 1-800-378-3854
Compuserve: Go Fuji

-Band Handheld Transceiver

Icom's ultra-slim IC-W32A dual-band transceiver fits comfortably in the hand and meets the needs of both experienced and novice operators. A total of 200 memories can be displayed by either frequency or name. Up to eight alphanumeric characters can be programmed for quick and easy identification on the LCD. Each band has 100 memory channels, and four DTMF memories have up to 16-digit capability.

You can receive both VHF and UHF bands simultaneously—and both frequencies can be viewed on the display—or use the V/V and U/U functions to receive two frequencies on the same band. You can transmit on either operating band. The IC-W32A has separate tuning and volume controls on the top panel to allow independent adjustment of each band. Icom's versatile VHF/UHF exchange function allows you to assign VHF/UHF tuning and volume to either knob. Frequency coverage is 144- to 148-MHz and 440- to 450-MHz transmit, and 118- to 174-MHz and 400- to 470-MHz receive.

The easy-to-use IC-W32A features a new guide function that allows quick description of the key function without searching a menu list. No function button means simple programming. Additional features include three levels of power output, multiple power-saver functions, large backlit keys, and full crossband and duplex operation. Scanning functions include programmed, full, and memory skip scan. The IC-W32A also has ten U.S. weather channels and built-in tone squelch.

The IC-W32A has a suggested retail price of $479.

ICOM AMERICA, INC.
2380 116th Avenue N.E.
Bellevue, WA 98009-9029
Tel: 206-454-8155
Fax: 206-454-1509

EN

CIRCLE 24 ON FREE INFORMATION CARD

Timid about getting on the. . .
World Wide Web?

You've heard about the *Information Superhighway* and all the hype that goes with it! Sort of makes you feel timid about getting on the Web. Put your fears aside! A new book, ***The Internet and World Wide Web Explained***, eliminates all the mystery and presents clear, concise information to build your confidence. The jargon used is explained in simple English. Once the tech-talk is understood, and with an hour or two of Web time under your belt, your friends will believe you are an Internet guru!

To order Book #403 send $6.95 plus $3.00 for shipping in the U.S. and Canada only to Electronics Technology Today Inc., P.O. Box 240, Massapequa Park, NY 11762-0240. Payment in U.S. funds by U.S. bank check or International Money Order. Please allow 6-8 weeks for delivery.

COMPUTER CONNECTIONS

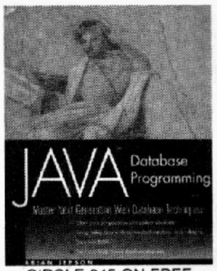
continued from page 22

capacitors. To add a shape to a drawing, you drag it from the stencil, and drop it in the work area. Intelligent connectors allow you to move shapes around and retain the lines connecting them. You can have multiple stencils and multiple drawing files open simultaneously. Each drawing file may have multiple pages. Pages may have backgrounds. You can create your own stencils and shapes. Shapes can have intelligence and customized data fields, so it's easy to create things like a bill of materials from a drawing. The overall package can be automated through OLE Automation.

Visio has diverged into two nonoverlapping subproducts, Visio Pro and Visio Technical. Both versions follow the same basic interface; they are distinguished by the stencils and wizards included. Visio Pro is for business analysts, software engineers, Web site developers, and database designers. Visio Technical is for everyone else; it is for diagramming electronic schematics and electrical and mechanical layouts. EN

NEW LITERATURE

continued from page 58



CIRCLE 345 ON FREE INFORMATION CARD

critical new Java database technologies and tools, including Sun Microsystems' Java Database Connectivity (JDBC) standard. It presents practical, step-by-step techniques with which you can harness the Java programming language. The book also explains how to create dynamic database applications and applets.

The book examines how Java pro grams access online databases. It sho how to integrate Java with networ database technologies. It teaches read how to program with JDBC and how develop JDBC drivers. The book a presents Java database tools and co libraries. The supporting Java Datab Programming Web site offers tinySQ generic and extendible SQL engine ten in Java; the tinySQL JDBC d and customizable Java database code