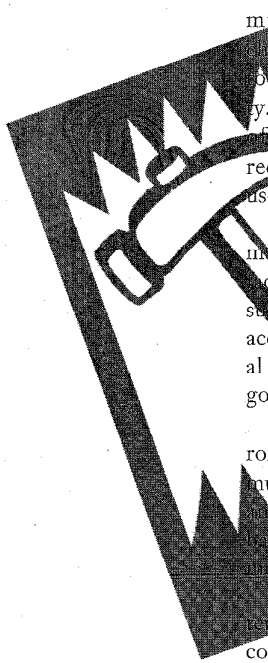
Meeting the Challenge of Software Maintenace

Taking the guesswork out of tool selection.

RECENT RESEARCH BY CASE ASSOCIATES Inc. shows that between 50 to 70 percent of a software engineer's time is spent making changes to mission-critical software. Hence, the tool with the greatest potential impact on the software organization is no longer the development tool, it is the maintenance and enhancement tool.

Today's maintenance efforts can best be described as system-replacement projects, in which organizations migrate or replace existing software functions rather than build them from scratch. Most of a maintenance programmer's time is spent figuring out legacy code: defining what the current system does, where, and how. After identifying what data the system uses and what processes it performs, the programmer must determine the impact of a proposed change. Reducing this code-analysis time is key to improving software-maintenance productivity. You need tools to improve the maintenance of existing systems by reducing the change-request backlog and responding more quickly to user needs.



Even well-run organizations, where software maintenance has not historically been a problem, face new pressure to change systems. This pressure stems from business-process reengineering, acquisitions and mergers, the rise of multinational operations, increased competition, and new government regulations.

Before migrating to a new computing environment such as client/server, your first priority must be gaining control of your current systems and requirements. You can then [illegible] on a stable, accurate, and [illegible] system.

Before we can gain control of our current systems, we must understand what causes the high costs and low productivity associated with software-maintenance practices. These problems stem from one or more of the following:

- poor system design and structure,
- excessive system complexity,
- limited system flexibility,
- limited or nonexistent documentation,
- inadequate project and process management,
- inadequate change and version management,
- inadequate release management, and
- inadequate maintenance tools.

MAINTENANCE CHALLENGES. As more and more new systems come on line, the pressure to continually adapt all of them in response to a steady stream of user requests has overwhelmed most software organizations. Today, even though most software engineers are assigned to current-system maintenance, they do not achieve the results expected of them. Change requests take too long and cost too much. The disparity between customer expectations and software-engineers' performance can be partly attributed to the misuse of software-engineering technology.

CASE has provided little to facilitate the change in emphasis from new-system development to current-system maintenance and enhancement

Traditional CASE tools and environments focus on developing a new system because when they were introduced in the 1980s that was what the market demanded. By the mid-'80s, CASE-tool vendors began claiming that the software-maintenance crisis could be corrected by *re*developing systems with CASE tools. These early tools and methods emphasized getting the requirements correct up-front — even though requirements change continually throughout a system's development, which can result in a system developed with CASE being judged unacceptable when finally implemented.

In short, although CASE has contributed to new-system development, it has provided little to facilitate the change in emphasis from new-system development to current-system maintenance and enhancement.

For example, CASE emphasizes logical design. Often, because of time pressures, programmers must make changes to the physical code but cannot reflect those changes at the logi-

Editor:
David Sharon
CASE Associates, Inc.
14915 SE 82nd Dr.
Clackamas, OR 97015
72204.1173@compuserve.com

0740-7459/96/$05.00 © 1996 IEEE

cal level. Integrated-CASE environments, which generate code, automate the links between logical design and physical code but do not resolve the issue of requirements management: the current system still requires continuous maintenance even as the new system is being developed. Thus, CASE tools that do not directly tie the logical model to the existing system are not a complete solution. Without this direct tie, it is difficult to verify and validate that the new, ICASE-built system actually replaces the existing system.

Even systems generated by ICASE tools must sometimes be changed at the code level, to accommodate security, database access, on-line access, and backup-and-recovery requirements. When these performance requirements are moved from the mainframe to a client/server environment, the tuning becomes more complex. In either situation, the programmer works directly on the source code generated by the ICASE tool. Any physical changes the programmer makes lack a logical-model equivalent, which causes disparities between the code and its logical-model counterpart, negating the ICASE tool's ability to accurately generate the application in the future.

ENGINEERING ALTERNATIVES. Reverse-engineering and reengineering tools strive to provide the link between the existing system and the replacement system. These tools can help the maintenance programmer. Reverse engineering, in particular, aids understanding of systems components and functional behavior. Reengineering focuses on extracting logical models from the current physical system, which then populate an ICASE repository for development of the replacement system.

Experience shows that it is labor-intensive to extract accurate models of the current physical system and modify them to conform to the metamodel of the target ICASE-tool repository. The incomplete linkage between the physical system and the logical models of the repository requires manual intervention and interpretation to complete. Reconciling the physical representations of the existing system with the logical-model requirements of the ICASE repository may take longer than it would to directly change the current system components. The key to success in reengineering involves gaining control and an understanding of existing systems. This comes from improved software-maintenance methods and tools.

TABLE 1
SOFTWARE MAINTENANCE: MAPPING THE PROCESS TO THE FACILITIES

Process Steps	Maintenance Facilities
1. Receive a change request	Change-request manager
2. Understand the change request	Program/System scanners Document management
3. Plan the change	Project management Change-request manager Impact analyzer
4. Figure out the code	Program/system scanner Program navigator Logic flow tracing Metrics analyzer
5. Determine change impacts	Program navigator Impact analyzer
6. Make the change	Language menus Language environment workbench
7. Test the change	Language environment workbench Test management Library link manager
8. Implement the change in new release	Release management Configuration management Library link manager

Although methodologists have developed formal methods for building new systems, they have ignored maintenance.

ORDER FROM CHAOS. The box on page 124 lists solutions to the problems maintenance programmers currently face. These programmers have always worked at the code level. Although methodologists have developed formal methods for building new systems, they have ignored maintenance. Because it involves working at the code level without formal methods, maintenance appears to be difficult to manage and control. In fact, the maintenance process is straightforward, as shown in Table 1. The keys for managing and controlling software maintenance are three-fold: Use a consistent set of tools, provide a mechanism for storing the programmers' discoveries, and formalize the process.

The ideal maintenance solution emphasizes understanding the current physical system and facilitating fast and accurate changes to it. The proper tools can improve maintenance productivity by helping the software engineer understand the current system and make changes and fixes more effectively. They also prepare the system for redevelopment. The facilities shown in Table 1

MENDING THE SOFTWARE-MAINTENANCE PROCESS

Although many in our industry believe that software maintenance is a complex and difficult task, the development of a comprehensive and consistent process is actually straightforward. Software organizations must solve most of the problems that follow. Each problem is accompanied by a proposed solution and a benefit for solving it.

Problem: Documentation is nonexistent or inconsistent with the physical system and is maintained as printed listings.

Solution: Store all related documentation in a database so that all who need to see it can access the documentation online. Establish links between the various documentation components: specifications, source code, report layouts, screen designs, database designs, file and record definitions, test cases, and so on.

Benefit: Eliminates the risk of missing important information.

Problem: Important information is buried in lots of text.

Solution: Break the system into components and allow those components to be viewed in parallel. Identify both the physical and logical components and cross-reference them so that the programmer can link and return to areas of interest. This achieves the automation of pencil notes, paper clips, and yellow stickies.

Benefit: Allows faster understanding of the system and its components.

Problem: It is difficult to understand the relationships between system components.

Solution: Store system components in a repository by scanning the code and defining the relationships between the discovered components.

Benefit: Eliminates errors resulting from not knowing the impact of change alternatives and, as a result of the traceability between related system components, verifies that all necessary changes have been made.

Problem: Discoveries made by one programmer are not saved and shared with others.

Solution: Develop a multiuser, multiaccess repository of system-component discoveries, versions, and changes.

Benefit: Discoveries need only be made once. New programmers will understand the system faster. All programmers are notified of a change when it occurs.

Problem: Excessive time is spent learning and figuring out the code.

Solution: Provide tools that trace logic and data flows and allow browsing within and among system components. Provide a diagrammatic representation of the program and systems structure as well as facilities that measure program metrics and identify complex and unstructured code. Restructure unstructured code as needed.

Benefit: The time to figure out the code and determine change impacts is reduced dramatically. The time to complete the change request can be determined with greater accuracy.

Problem: Difficult to determine the business and system rules embedded in the source code.

Solution: View the program logic with the I/O by breaking the source code into code blocks that represent database-access code, screen-access code, conditional logic, and operational logic.

Benefit: The business and system rules can be isolated, understood, and recorded for easier maintenance.

Problem: Programs contain redundant information that makes it more difficult to understand a system and make changes.

Solution: Provide tools that identify logic paths not followed (dead code) and that look for modules with similar logic-path structures. Many redundant modules are not identical because they were written by different people, but they perform the same functions and use the same data.

Benefit: Programs are made cleaner and easier to understand.

Problem: There are many physical representations of the same logical data, making data usage difficult to understand.

Solution: Provide tools to identify candidates for rationalization through consolidation into single copybooks.

Benefit: Data usage becomes consistent. Data tracing becomes easier, making programs easier to understand.

Problem: Testing lacks comprehensiveness and consumes too much time, especially test-data creation and management.

Solution: Provide test data as a reusable component stored in a repository along with its relationship to the system component to be tested. Provide coverage-analysis tools to ensure all paths are tested.

Benefit: Tests become consistent, thorough, and efficient.

Problem: There is no comprehensive set of integrated tools.

Solution: Provide a tool set that focuses on understanding and maintaining current systems, along with a repository to store all the programmers' discoveries and system components.

Benefit: Gains control of current systems, reduces change backlog, and improves productivity.

Problem: Control procedures are not applied to maintenance.

Solution: Provide configuration management to maintain multiple versions, the differences between versions, and synchronization and control of reusable system components between projects. Provide process management to ensure the correct procedures are followed and the correct tools used. Make project documents readily available for status checks, quality reviews, and project schedule predictability. Provide release management to identify which system components and versions form a specific release and to verify that all components are available with all the necessary checks.

Benefit: Provides control of current systems, which results in improved quality, accuracy, and predictability.

manage and control all system-component and project-level data and the relationships between the data. They also promote information sharing and reuse between the maintenance process steps. The core components include process management, which ensures that the correct procedures are followed and the correct tools used; project management, which sustains a consistent work-breakdown structure and communication between project-team members; configuration management, which maintains multiple versions, the differences between versions, and provides for synchronization and control of reusable system components between projects; and the repository, which stores system components, versions, and changes. The maintenance facilities directly address and solve the causes of high cost and low productivity in software maintenance.

THOSE WHO CAN'T, MAINTAIN. Traditionally, talented programmers have followed their own variation of the saying, "Those who can, do; those who cannot, teach." In the case of programmers, though, the less capable have been relegated to maintenance tasks while the talented have been given the more challenging work of developing new systems. However, as systems become more complex and the technology incorporated in them becomes more obscure, the need for talented maintenance experts grows. Today it is not unusual to find the best and most experienced programmers maintaining mission-critical systems. They reap many rewards for doing so. Well-maintained systems

As systems become more complex, the need for talented maintenance experts grows.

- allow for better working practices, which increase job satisfaction;
- improve customer service, reduce costs, and increase productivity;
- bestow prestige on those associated with contributing to the organization's success; and
- provide for more efficient upgrades and a smoother migration path.

The increasing burden of software maintenance requires that we dedicate our best programming minds to the task. If we must do so, it is only fair that we arm them with the best tools available and provide them with clear, coherent software-maintenance procedures. ◆

David Sharon is the president of CASE Associates, Inc., a firm that specializes in software-engineering process quality and tool evaluation, selection, and deployment services.

TOP DRAWER

Internet programming language. Java is a general-purpose language that is object-oriented, distributed, interpreted, secure, architecture-neutral, portable, multithreaded, and dynamic. Java supports programming for the Internet in the form of platform-independent Java applets. It uses a simplified version of C++ that omits such rarely used and confusing features as operator overloading, multiple inheritance, and extensive automatic coercions. Object orientation and automatic garbage collection make programming easier. Portability is achieved with a Java compiler that generates ANSI C runtime code with a clean, Posix-based portability boundary. This architecture-neutral code lets Java applets move freely across the diverse platforms of the Internet. Many Web browsers and Web publishing systems — including Netscape's Navigator, Macromedia's Shockwave, and Microsoft's forthcoming Blackbird — offer direct or indirect support for Java applets. The current beta version of Java is available at no cost from Sun Microsystems' Web site at http://java.sun.com/starter.html. Contact Sun at (415) 960-1300; fax (415) 786-7546.

READER SERVICE 90

Internet browser. Netscape's Navigator 2.0 provides Web exploring, e-mail, newsgroups, chat, and FTP capabilities in an integrated package. It supports Live Objects and other interactive multimedia content such as Java applets, frames, and Netscape inline plug-ins. Along with several end-user improvements, version 2.0 provides many developer enhancements, including frames, which provide simpler navigation and greater flexibility in Web-site design and let developers divide each page into regions; Inline Plug-ins, which integrate existing technologies such as Apple QuickTime movies, Adobe Acrobat documents, and Macromedia Director presentations; Java applets, powerful and secure interactive objects that can enable animation, live updating, and two-way interaction; and JavaScript, a cross-platform language based on Java that extends the programmatic capabilities of Netscape Navigator 2.0 to developers of all experience levels. Also available is Netscape Navigator Gold 2.0, a premium Internet client that adds integrated WYSIWYG document creation and publishing capabilities to Navigator 2.0's standard features. Netscape Navigator 2.0 costs $49; Navigator Gold costs $79. A subscription, which provides a year of upgrades and the appropriate version if you switch to a new platform, costs $17. Contact Netscape at (415) 528-2555; http://home.netscape.com.

READER SERVICE 91

Internet multimedia integrator. Macromedia's Shockwave lets designers integrate into their Web sites interactive multimedia presentations created in Director 4.0, which adds synchronized graphics, sounds, animation, and local interactivity to HTML's static text and graphics. Shockwave is currently available as a plug-in for Windows users of Netscape Navigator 2.0 Beta 3; versions for the Macintosh and other browsers are planned. The current beta version of Shockwave is available for Windows 3.1 and Windows 95 at no charge by accessing Macromedia's Web site at http://www.macromedia.com/Tools/ Shockwave/index.html. Director 4.0 retails for $1199. Contact Macromedia at (415) 252-2000; fax (415) 626-1502.

READER SERVICE 92

Web script language. htmlscript is Volant's browser-independent language for Web servers. Installed as a standard GCI application, it provides multiple administrative controls, can be mixed freely with HTML in a file, and ensures browser independence by using a preprocessor to convert htmlscript tags to HTML code. The tags provide the capability to do logic, calculations, and file I/O operations. htmlscript is available for most Unix and Unix workalike systems, with Windows NT and Sun Java versions planned. An introductory, 500-user license costs $99 and includes a year of free updates. Contact Volant at (619) 490-2570; fax (619) 490-0548.

READER SERVICE 93

Visual-language builder. DV-Centro is a C++ framework for building visual-language systems that has been used to create visual-language applications such as diagram editors, finite-state-machine editors, and CASE tools. DV-Centro provides several graphics classes, including primitives, rendering, properties, and transformations; graphical-constraint management for graphical connectivity and containment; event-handling classes for platform-independent definition of events, states, and responses that define sytsem behavior; mechanisms for graphical editing, including *n*-level undo/redo, copy/cut/paste, and save/restore; run-time-checking and memory-management utilities; and Python bindings to speed development and compile times. Available for SunOS, Solaris, and HP-UX platforms, DV-Centro pricing starts at $15,000. Contact DataViews at (413) 586-4144; fax (413) 586-3805.

READER SERVICE 94

Design automation tool. Statemate 6.0 lets system engineers build complete executable models of a real-time system — even before the hardware and software have been partitioned — to validate its behavior and funtionality. Designers use Statemate 6.0's Statechart state diagrams to create an executable specification, a virtual prototype whose specifications can be changed on-the-fly. Dynamic tests can be applied to complex systems, letting users check for deadlocks, racing conditions, and nondeterministic behavior. Once the graphical model is designed and validated, Statemate automatically generates the complete code, in either C or Ada, to represent the system model, fully preserving its behavior. Statemate 6.0 is available for SunOS, Solaris, HP-UX, IBM, AIX, and Digital VAX/VMS workstations. Cost per seat starts at $25,000. Contact i-Logix at (508) 682-2100; fax (508) 682-5995.

READER SERVICE 95

Change-request management system. Atria's ClearTrack is a customizable client/server change-request management system. ClearTrack lets teams record change requests, track their evolving status, and obtain metrics using its query and reporting facilities. Its features include predefined states and transitions and queries and reports, e-mail integration, and a GUI. ClearTrack's explanatory change-request attachments include sample output, core dumps, and bit maps. It also integrates with Atria's ClearCase software-configuration management system. ClearTrack is available for several Unix platforms, including SunOS, Solaris, HP-UX, and SGI IRIX. Support for Windows NT, Windows 95, and other Unix platforms is planned. A first single-user ClearTrack license costs $795; bundled with ClearCase, it costs $4,595. Contact Atria at (617) 676-2400; fax (617) 676-2550.

READER SERVICE 96

Windows automated test suite. SQA Suite for Windows 95 and Windows NT provides integrated testing for enterprise client/server applications developed for those two operating systems. The 32-bit implementation's user interface has a Windows 95 "look and feel" and installation style, has the same functionality as other versions of SQA Suite 4.0, and adds evolutionary extensions to SQA's Object Testing technology. The suite also helps users migrate 16-bit Windows 3.1/3.11 applications to 32-bit Windows 95 and Windows NT versions. SQA Suite for Windows 95 and NT sells for $2995; pricing for the five-agent client/server version starts at $12,395. Contact SQA at (617) 932-0110; fax (617) 932-3280.

READER SERVICE 97