



Industrial-Strength Management Strategies*

NORM BROWN, Software Program Managers Network

As our industry undertakes ever larger development projects, the number of defects occurring in delivered software increases exponentially.

Drawing on his experiences in the defense industry, the author offers nine best practices to improve the management of large software systems.

The United States Department of Defense spends an estimated \$42 billion annually for the development and maintenance of its computer systems; only \$7 billion of this sum buys hardware.¹ Software has become a major cost, schedule, and performance driver of virtually all DoD weapons, command-and-control, and information systems.² As of July 1995, the DoD had more than \$256 billion under contract for these systems. Although that figure applies to entire systems, including software, it has become alarmingly apparent to DoD executives and the US Congress that the software tail wags the system dog. When software delays the fielding of a major system, the costs can become enormous.

* Copyright © 1996 Institute of Electrical and Electronics Engineers. Reprinted from "Industrial-Strength Management Strategies." *IEEE Software* 13, no.4 (1996): 94-103.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the Software Program Managers Network's products and services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank e-mail message to: info.pub.permission@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.



These problems are not unique to defense systems. The Denver Airport Baggage Handling System, the Federal Aviation Administration's Advanced Automation System, and the English Channel Tunnel are just a few of the systems hit with substantial economic consequences attributable to problems in developing large-scale software.

Many large software projects underway now will experience significant problems or even be canceled after a very substantial investment. The number of defects occurring in software delivered to the field is not a linear function of size: defects increase exponentially. Capers Jones has determined that defect potential is directly related to the function point total of an application raised to the 1.2 power. As software systems become larger, defects become a much more significant problem.³

Efforts to both identify and eliminate defects, and to prevent their introduction, must be a major component of any strategy to reduce the cost and risk of software development. In many cases, the true nature and pervasive extent of underlying project problems remains invisible to the project manager until it is too late. There are no air bags for software projects.

As large-scale software systems have become commonplace, the ability to effectively manage these systems has not kept pace. Development costs continue to rise while productivity declines. Virtually all other nations with highly qualified software practitioners have development costs between \$125 to \$250 per function point, except for Japan at \$1,600 and the US at \$1,000. This difference in per-function-point cost is partially a result of the disparity in labor costs among these developed nations. Software practitioners in some countries often perform the same functions as their American and Japanese counterparts for a fraction of the salary. System development costs have been increasing exponentially with size.

Management of system development has been found to be a key determinant of cost. If American software managers do not improve how their large systems are managed, they will likely see their nation's software business move overseas, as already happened to their shipbuilding industry.

Although this article draws on my experiences in the defense industry, I present here a distillation of industry-wide techniques for managing successful large-scale software projects.

MANAGING TO FAIL

Watts Humphrey and the Software Engineering Institute have accomplished much in the last 10 years to alert the software industry to the need for process improvement. Many other individuals and organizations have been involved in bringing about such improvement. Yet my experience with many software organizations in government and industry, reinforced by industry data, indicates that organizational improvement has been occurring at a glacial rate. According to the SEI, of the 379 organizations at 99 companies that are advanced enough to have process improvement programs in place and that have conducted SEI maturity assessments, 73 percent don't rate higher than level one. Discussions with many managers of large-scale software projects in both DoD and industry indicate that they need directly useful strategies, practices, and techniques to substantially accelerate the pace of process improvement.

Root problem. In 1987, the Defense Science Board, under the chairmanship of Fred Brooks, sought to determine the underlying causes of large-scale software problems — the well-known cost overruns, schedule slippages, and performance problems. Not surprisingly, they concluded that

If American software managers do not improve, they will see their nation's software business move overseas.

"Today's major problems with military software development are not technical problems, but management problems." The Board noted that "Many previous studies have provided an abundance of valid conclusions and detailed recommendations. Most remain unimplemented."⁴ The Department of Defense, despite its deep dependence on software for virtually all its systems, has been slow to address the thorny problem of poor software-project management, ever pursuing instead the software silver bullet *du jour*. The DoD is not alone in this quest, which is also common in the commercial software industry. Of course, there are no silver bullets, and chasing them only postpones real improvement.

Assessment of many large-scale software projects suggests that the largest single cause of poor software management is poor understanding of essential management practices and techniques, of what works and what doesn't. This lack of understanding is echoed in what I call the "senior-management problem." Corporate or government executives responsible for software often don't understand the critical aspects of large-scale software development. Management's lack of understanding leads to unwillingness to support the needed infrastructure (such as training and tools) and process improvements. It also results in failure to effectively support the entire software team, con-



tractual and other actions that erode the team's credibility and processes, and unrealistic commitments in time and money.

This senior-management problem is compounded by the prevailing senior corporate and government management mindset. These organizations aren't aware that large-scale software management can benefit from the same engineering-management techniques successfully used to manage large-scale hardware projects.

Too many projects have failed and many more are failing because of counterproductive management. A cultural change in software management is imperative, and although bringing this about is not easy, accomplishing it is vital to the software industry's competitiveness.

Software Program Managers Network. By 1992, the DoD's failure to make needed improvements in its software processes and its neglect of the software arena generally had left DoD software management in terrible disrepair. In reaction, a group of US Navy program managers and software practitioners formed the Software Program Managers Network to identify what worked and what didn't in real-

it has begun providing support teams to large-scale government projects. The Network seeks to avoid bureaucratic entanglements and bottlenecks, to remain responsive to the project management needs of the software community. Senior Service and DoD officials have recognized the Network's utility, encouraged its activities, and become members themselves. The Network is not confined to the shores of the US, however. It is expanding daily to include non-US members.

BEST PRACTICES INITIATIVE

In July 1994, the two officials responsible for virtually all DoD software development established the department-wide Software Acquisition Best Practices Initiative to "improve and restructure our software acquisition management process," and to "expand and support the efforts now underway by the Network to identify and convey these practices." Noel Longuemare, then acting under secretary of defense for acquisition and technology, and Emmett Paige, Jr., assistant secretary of defense for command, control, communications, and intelligence — themselves Software Program Managers Network members — described the need for the initiative as follows:

Current DoD software acquisition practices have not proven to provide an effective framework for managing the acquisition of large-scale software development and maintenance programs that are an essential part of increasingly complex weapons systems.

Considering that the DoD is about as top-down hierarchical an organization as can be found, one of the most impressive aspects of this initiative is its "bottom-up" focus on program managers deciding the appropriate change for their organizations in the context of

their particular program and environment.

The Initiative has four primary purposes:

- ◆ to focus the Defense acquisition community on employing effective, high-leverage software-acquisition management practices;

- ◆ to let program managers focus their software management efforts on producing quality software, rather than on activities directed toward satisfying regulations that have grown excessively complex over time;

- ◆ to let program managers exercise flexibility in implementing best practices within disparate corporate and program cultures; and

- ◆ to provide program managers and staff with the training and tools necessary to effectively use and achieve the benefits of these practices.

Three main components comprise the Initiative. First, a "guru group," the Airlie Software Council, identifies processes and suggests solutions that can facilitate large-project development and maintenance. Its findings include the nine Principal Best Practices described later.

Second, seven Issue Panels, staffed equally from industry and government by 180 software practitioners, address the areas of risk management, program planning and baselining, program visibility, program control, engineering practices and culture, process improvement, and contracting and solicitation. These panels reviewed 163 practices submitted by 56 companies. From these and their members' own experiences on large-scale programs, the panels identified 43 supporting Best Practices that not only support the nine Principal Best Practices, but also provide insights into advanced management and development techniques.

Third, the Program Managers Panel, a group composed of highly experienced and successful industry program managers, provide a program manager's

One of the most impressive aspects of this initiative is its "bottom-up" focus.

world software projects, and to help success happen by conveying this information to busy program managers.

The Network and its activities have grown substantially over the last four years, and it now encompasses US commercial and academic sectors. Recently

perspective and real-world anchor.

In the interest of assuring robust and meaningful results, the Airlie Council and Issue Panels checked and verified each other's outputs. The Program Managers Panel performed a "sanity check" on the outputs of both the Council and the Issue Panels. All practices underwent an extensive review and comment process.

THE AIRLIE SOFTWARE COUNCIL

Composed of some of the software industry's leading experts, the Airlie Software Council is charged with identifying fundamental processes and proven solutions that are essential to successfully managing large-scale software development and maintenance. The Council was purposely structured to include highly successful managers of large-scale software projects, internationally recognized authors, prominent consultants, and executives responsible for software development at major companies. Council members include Vic Basili, Grady Booch, Norm Brown, Christine Davis, Tom DeMarco, Mike Dyer, Mike Evans, Capers Jones, Tim Lister, John Manzo, Lou Mazzucchelli, Tom McCabe, Frank McGrath, Roger Pressman, Larry Putnam, Howard Rubin, and Ed Yourdon. Considering this diversity, opinions on the issues were expected to be contentious at least, if not irreconcilable. Remarkably, only strong consensus emerged.

From the beginning, the Council has focused on identifying the activities that have the greatest effect in achieving successful management of large-scale software development — the high-leverage payoffs. The Council has explicitly avoided silver bullets, and has identified and articulated

- ◆ nine Principal Best Practices for software management;

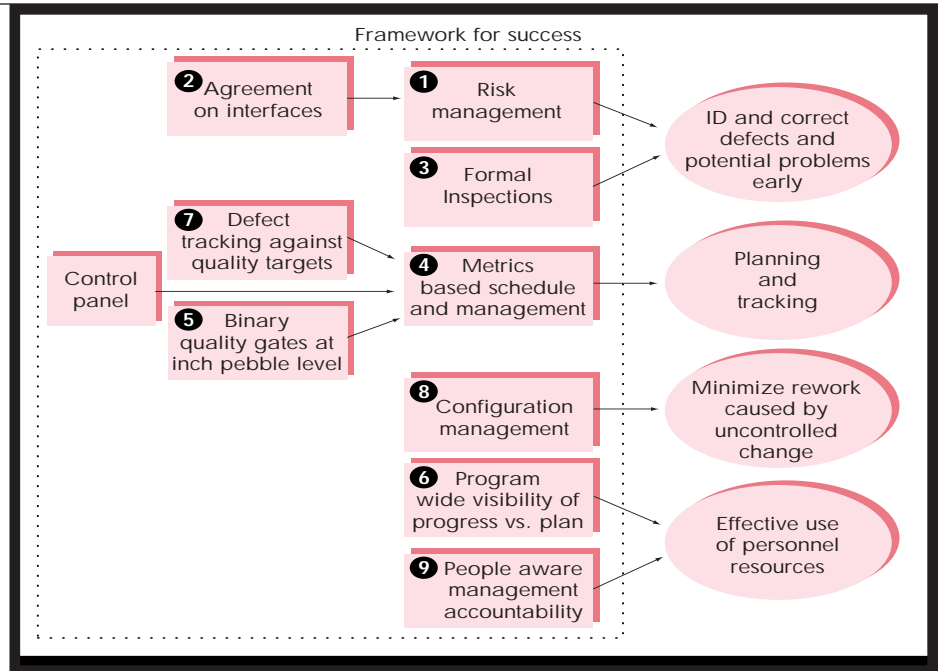


Figure 1. The nine Principal Best Practices constitute a control mechanism that provides a disciplined set of techniques for containing project complexity.

- ◆ a Software Project Control Panel that provides the project manager and staff with key indicators of project status, in three layers of increasingly detailed metrics;

- ◆ a Little Yellow Book whose questions provide a means for program managers to assess how well project management understands and addresses the key issues involved in conducting a successful project;

- ◆ a set of software project caveats — some "worst practices" to avoid;

- ◆ a set of quantitative targets — project stretch-goals and malpractice warning levels;

- ◆ a project "breathalyzer" test to help determine if a project is fit to be "on the road."

THE SOFTWARE MANAGEMENT FRAMEWORK

Experience within DoD has shown that large-scale software projects have a high inherent complexity, which can quickly grow into chaos when not effectively controlled.⁶ To contain this complexity, the tasks associated with these factors must be properly carried out. The nine Principal Best Practices con-

stitute a control mechanism that addresses these factors, which Figure 1 shows, and provide a disciplined set of techniques for containing project complexity.

Identify and correct defects and potential problems early. An adage from the construction business, "Curing concrete waits for no man," speaks to the importance of removing defects and correcting problems before the concrete sets up. This rings just as true for software. It's also much more cost-effective to identify and fix defects in construction drawings than to disassemble and reconstruct the forms. In our software analog, costs to correct defects also increase as a defect propagates from one project phase to the next.

Extensive rework caused by failure to identify and fix defects at their source contributes greatly to project overrun and schedule slip. The cost of rework on large-scale projects has typically consumed 40 to 50 percent of the total software development budget.⁷ All work products must be reviewed to identify defects, and corrective action taken immediately. Such early intervention obviously applies to any development methodology, not just a water-



fall development, and to whatever phase a defect is found in.

Undetected defects are difficult to correct. This may be obvious, but many project staffs either don't know how to effectively identify defects or have decided not to for various reasons, including running up the project's bill. Current government contracts typically pay contractors by the staff-hour, which provides little incentive to reduce expensive rework.

It is important to minimize the time between "insertion" of a defect and its discovery through inspections, divine inspiration, testing, and so on. A recent study indicated that for large military software projects, the average time-delay is nine months, which means that by the time the defect is identified, nobody can remember what they were doing when the defect was introduced.

You should be concerned not only with correcting the current project, but also with correcting the processes so that both the current project and future projects will benefit. Corrective action entails not only correcting specifically identified defects, but also examining and fixing the defect-causing mechanism. Such improvement is specifically

you've made in getting there without objective measures. For large-scale software, measuring progress against a planned set of accomplishments delivers an effective measure of progress achieved — termed "earned value." It is essentially impossible to build successful large-scale software without detailed planning of all activities needed. That planning must be based on reasonably accurate estimates of resources and time.

Minimize rework caused by uncontrolled change. Uncontrolled change causes chaos. Typically, the larger and more complex the project, the greater the uncontrolled change and chaos. Too often chaos and its pernicious effects go unrecognized until too late. Uncontrolled change comes from all development efforts, including requirements, deliverables, and external interfaces. Requirements creep is another significant contributor to chaos, one that can seal the fate of a program if not effectively controlled. Systematic use of problem reports, change control, engineering change proposals, and cost and schedule baselines is essential to controlling change.

Make effective use of your people. The single most important determinant of a project's success is the ability, experience, and motivation of its people. There is a very significant payoff from retaining and motivating competent people experienced in the management and technical methods appropriate to the application being developed. The big challenge is in doing it.

NINE PRINCIPAL BEST PRACTICES

Although the following practices are useful individually, their complementary nature provides a strong synergetic effect when they are used as an inte-

grated set. Using them will not guarantee success, but they can help facilitate it and avoid failure.

Those familiar with process improvement models such as the Capability Maturity Model⁹ will quickly realize that these practices supply tactical solutions to the model's strategic orientation. The practices map to many of the model's Key Process Areas, and should assist organizations striving to advance to the next CMM maturity level.

These practices are not rocket science. They are readily implementable. Although some practices may require training in basic skills, such as conducting effective meetings as a necessary foundation for formal inspections, they can, for the most part, be implemented without making investments in new equipment, technologies, or staff.

Practice 1. Formal Risk Management.

Rear Admiral Bill Carlson, United States Navy, served much of his senior-officer career managing large and complex combat systems. He is well-known for his philosophy that "If you're not managing risk, you're managing the wrong thing." From their own experience, the Airlie Software Council members came to a similar conclusion. Of the nine practices, formal risk management is first among equals.

All software development has risk. The cost of resolving a risk is usually relatively low early on, but increases dramatically as the project progresses. Notwithstanding all this, many assessments of DoD and industry reveal that risk is often given only lip service. In the council's opinion, the absence of a comprehensive risk management program is a leading indicator of incipient project failure. Effective risk management requires acceptance and decriminalization of risk as a major consideration for software program management; commitment of program resources; and use of formal methods for identifying, monitoring, and managing risk.

**Rear Admiral
Bill Carlson:
"If you're not
managing risk,
you're managing
the wrong thing."**

covered under the level 5 Defect Prevention key process area of the SEI's Capability Maturity Model.⁸

Plan and estimate. Although you may reach a destination without a map, you can't quantify how much progress



Essential Elements. A senior team member should be assigned responsibility for, and be formally designated as, the project's Risk Officer. He or she should

- ◆ Identify risk. Risk management should include risks over the full life-cycle. A database should be compiled for all non-negligible risks, including cost, schedule, technical, supportability, and programmatic risks. Each risk item should be characterized by probability of occurrence and by consequence.

- ◆ Plan for and mitigate risk by keeping risk items off the critical path through early identification and workarounds. Use mitigation efforts to avoid "single-points-of-failure" wherever possible. The number of unresolved risk items should be tracked on the critical path, along with the probable cost for unresolved risk versus risk reserve remaining.

Plan risk-item resolution as early as possible. Include a calculated risk reserve of time, money, and other key resources to deal with risks that materialize unexpectedly.

- ◆ Provide frequent risk status reports to the program manager, including: the top ten risk items; unresolved risk items on the critical path; the number of risk items resolved to date, new risk items since last report, and unresolved risk items; and the probable cost for unresolved risk versus risk reserve.

- ◆ Maintain an anonymous reporting channel by emplacing mechanisms to ensure anonymous means for staff to identify risks and problems. Ensure that the project's software development culture is open and nonthreatening. Help foster a "no cover-up" in the organization.

- ◆ Decriminalize Risk. Although risk management and associated risk reserves might seem a good idea to many managers, some organizational environments constitute a de facto obstacle. Problems in such organiza-

tions range from a "controller staff" view that a "pot of risk reserve dollars" is too good to leave on the table, to a management view that "even merely identifying a potential risk is equivalent to its having happened." In such instances, a project manager must identify and manage risk in more subtle and creative ways.

Practice 2. Agreement on Interfaces. Although system interfaces typically constitute essential elements of a system's requirements and architecture, these interfaces are often beyond the control of the project manager. The typical consequence of not achieving early stabilization of external interfaces is necessary fixes later, which can result in significant program redesign, recoding, and retesting — all extremely expensive and time-consuming.

Essential Elements. System interfaces must be defined before software analysis and design. User interfaces and system external and internal interfaces are all important. Rapid prototyping of the graphical user interface should be used as a tool to define user requirements.

The user interface should be fully identified and baselined before beginning development. It should be part of the system specification and should include definitions for screen fields, each input/output data item, and navigation between screens. For embedded systems, a separate system specification for the software should be prepared.

Only the actual operational user can define and accurately verify the user interface's correctness.

Practice 3. Formal Inspections. Rework done to fix defects not found when they are introduced accounts for between 40 and 50 percent of total development costs for DoD-sized software. Add to this that the cost of fixing a defect increases significantly each time a defect propagates to the next

Formal inspections and even walkthroughs are generally more effective than formal reviews of the same work product.

phase, and you've got a very costly problem that grows the longer defects remain hidden and uncorrected. Although such rework can be cut drastically, not many large-scale projects seem to know how. Formal inspections such as Fagan inspections¹⁰ are the solution. They typically find 80 percent of defects as they happen, compared with walkthroughs, which typically find only 60 percent.

Accordingly, formal inspections, because of their effectiveness in identifying defects and potential problems early, can have a resounding effect on reducing rework, particularly when they are used frequently. According to Capers Jones, each defect-removal operation — review, inspection, and test — will find about 30 percent of the defects present.³ Thus, achieving high levels of quality requires multiple defect-removal operations. Jones has found that "best-in-class" software groups use more than 10 stages of defect-removal operations. These include full design reviews, code inspections, document edits, and multiple testing stages. Formal inspections and even walkthroughs are generally more effective than formal reviews of the same work product. Such reviews tend to involve lots of people in highly unproductive ways that achieve little. Defects can be found early if formal inspections, as a process, are applied



not only during coding or testing, but during requirements definition, architecture, and design efforts.

Essential Elements. Use small teams of prepared reviewers with assigned roles. Ensure that entry and exit criteria exist for each review. Use formal inspections early in all development phases, ensure user participation in them, and make sure their specific “rules” are fully understood. Train people to effectively conduct inspection meetings. Finally, track the duration of inspection sessions, defects identified per session, the duration-to-defect ratio, and the duration from making the defect to finding it.

Practice 4. *Metric-based scheduling and management.* This practice aims to identify potential problems early. As far as the project manager is concerned, early identification of problems is the only reason for putting adequate and effective metrics in place. To achieve success in software development projects, detecting potential problems early is essential because the cost to fix a problem rapidly increases with the amount of time it remains uncorrected. Comparing earned value with the actual costs of the tasks completed provides

Establish an anonymous communications channel anyone can use to report problems to the project manager.

a fundamental indicator of problems developing in this area.

Project management metrics are the yardstick for measuring actual progress

achieved against the project’s baseline plan; they become a warning indicator for potential problems and assist in localizing problems. For example, we must determine if an indicated schedule delay is the fault of the project staff or if the plan was too aggressive to begin with. A highly productive and effective development staff should not be punished for failing to meet unrealistic objectives.

Cost and schedule estimates should be based on empirical data. Metric-based planning requires early calculation of size, projection of costs and schedules from empirical patterns, and tracking of project status through the use of captured result metrics. The program should estimate cost and schedule using data from completed projects of similar size and objective. These estimates should be compared with a cost model estimate that uses historical data.

Essential Elements. Plan short-duration tasks with measurable products. Estimate cost and schedule by analogy with past completed projects, then compare these with cost model estimates.

Review key earned-value indicators at frequent intervals throughout the project. Monitor and take action based on the cost-performance index and the to-complete-performance index. Monitor and manage defect closure time. Track control panel metrics to identify potential problems before they become major sources of rework. Don’t hide problems with rebaselining. Manage defect closure time.

Practice 5. *Binary quality gates at the inch-pebble level.* The adage, “the devil’s in the detail,” holds especially true for software. When project planning and monitoring are based on insufficient detail, discussion of where the program is and how it is progressing becomes illusory. Program management without

detailed planning and clear progress measurement might just as well be called “let’s pretend.” By pushing project efforts to fine-grain resolution, specific development activities can be far more effectively identified, planned, and tracked.

Planning must be based upon an “inch-pebble” detailed activity network — each task is no more than 5 percent of the duration and effort of the release, and is performed at least 95 percent by a single, lowest level organization. The product developed by the task must be of a single type.

Employing binary quality gates — objective acceptance criteria and tests for determining if output product is acceptable and that don’t allow efforts to be considered complete until they pass all their predefined acceptance criteria — lets effective and meaningful evaluation of actual progress be made against the plan.

Essential Elements. Ensure that development tracking is based upon actual products. Every lowest level task should be of short duration, should expend a small percent of the total development budget, should produce a tangible product necessary for a required deliverable, and should have a binary quality gate defined for it. No earned value credit should be given for a task until the task passes the binary quality gate.

Practice 6. *Program-wide visibility of progress vs. plan.* This practice aims to get the entire project staff actively identifying problems and risk. When everyone is involved, the likelihood of missing problems is greatly reduced. Such broad involvement substantially strengthens risk management and increases the likelihood of program success.

Essential Elements. Make the control panel accessible to all team members

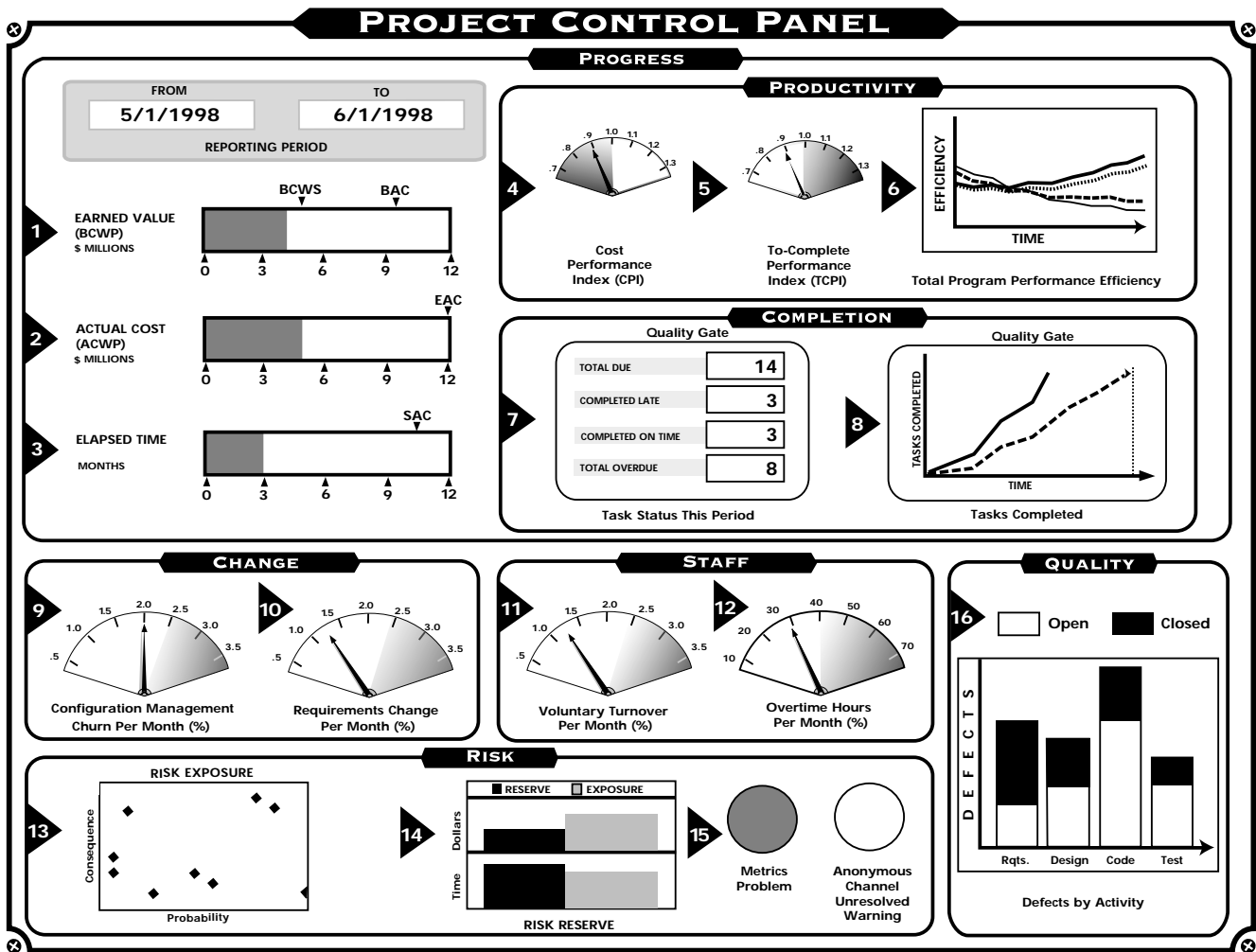


Figure 2. The software project control panel. Similar to an aircraft pilot's control panel that provides critical information regarding the aircraft's status, the software project control panel displays current project status.

and the customer. Establish an anonymous communications channel anyone can use to report problems to the project manager. Although this will also be a channel for malcontents, it is much better to get false alarms than to miss a major problem. Top-down program-wide visibility will reduce the number of reports of nonproblems.

Practice 7. Defect tracking against quality targets. Perhaps not surprisingly, many programs track neither the source of defects nor their project's performance over time on some basic indicators relating to defects, schedule, cost, requirements, documentation, and staff.

The "software defect snowball effect" is a significant contributor here: The bigger the software gets, the greater the rate of defects occurring.⁷ Typically, 20 percent of all software modules contain 80 percent of the defects.

Defects should be tracked formally at each project phase. Use configuration management to record and trace each defect through to removal. In this approach there is no such thing as a "private defect," that is, one detected and removed without being recorded.

Essential Elements. Implement practices to find defects when they occur.

Establish a goal for delivered defects, including requirements problems, per unit of size. Use configuration management to report and track all defects found through all techniques. Also track the average and maximum time to close a defect after it's reported. Monitor the organization's defect removal efficiency by tracking the number of defects found and fixed during development and during the system's first year in the field.

Practice 8. Configuration management. Not controlling change, particularly in a team development of software, dramatically increases complexity



to a level approaching chaos, and to almost certain failure. This practice is based upon two simple rules:

- ◆ Any piece of information that is used concurrently by more than one individual or organization should be controlled through a project configuration-management process.

- ◆ Any piece of information approved at a quality gate should be controlled through project configuration management.

In essence, good configuration management is a formal process for controlling shared products and all changes to a baseline, and to formally track more than deliverables.

Essential Elements. Use change tracking to formally track the status of shared products, problem reports, cost-and-schedule baselines, case files, internal and external interfaces, reports used by more than one organization, and all concurrently nonbaselined information shared within the program or approved for internal release through binary quality gates.

Automation is essential, because of the amount of information to be controlled and its dynamic nature. Ensure configuration identification by placing shared information under project-level control and identifying each controlled

Automation is essential, because of the amount of information.

item by version ID. Monitor status and maintain records describing history, content, and release records for products controlled by configuration management. Continually evaluate information under control for content, baseline integrity, and change status.

Practice 9. People-aware management accountability. This practice is intended to keep a spotlight on people as the crucial foundation for program success. The single most important determinant of a project's success is the quality, experience, and motivation of its people. A significant part of software development and maintenance requires human intellect and creativity at a level that greatly exceeds that required for most other jobs. Watts Humphrey's Personal Software Process may represent a significant breakthrough in increased personal productivity and demonstrates the effect individuals can have on project outcome.¹¹

No matter how well versed someone is in the relevant technology, a substantial investment is still required to give that person a detailed understanding of the specific application being developed or maintained. With the rapid advances in software technology that have and will continue to occur, a high percentage of software professionals are not current in the best technology related to their job. Demands upon intellect and creativity are much greater with such new technologies as client/server networks and the abstractions required for object-oriented analysis and design.

Despite efforts to fully document software, some vital information about a project exists only in the minds of select individuals. Accordingly, the project manager must assume explicit responsibility for addressing both staff quality and voluntary turnover rate. What this really means is that the preceding eight practices will be of little help if the project's technical staff is unqualified or transient.

The top two jobs in the United States, according to *Money* magazine, are computer engineer and computer systems analyst.¹² Studies of large projects have shown that 90th-percentile software teams typically outperform 15th-percentile teams by factors of 4 or

5 to 1, with individual productivity ranges of 26:1.

Studies have shown two relevant relationships:

- ◆ a high correlation between organizations that invest in timely training and their developmental success, and

- ◆ a high percentage of software professionals do not keep up with technology advancement.

Investing in the right kind of just-in-time training is not free, but appears to be essential for successful programs. Encouraging technical currency comes at a much lower cost because techniques such as brown-bag seminars, technical-journal subscriptions, and videotapes are not big-ticket items.

A key to effective use of personnel, according to John Manzo, an Airlie Council member, is to establish an open and empowering environment based upon truth and trust. The project manager must be rewarded and held accountable for staffing qualified people — those with domain knowledge and similar experience in previous successful projects — as well as for fostering an environment conducive to high morale and low turnover.

Essential Elements. Minimize burnout. Extended periods of work greatly in excess of 40 hours per week often lead to excessive turnover later. Keep records of actual hours worked as well as hours charged to the customer. Help your staff maintain their personal technical currency. Invest in timely training of project staff.

Treat your stars well. The near-term and projected long-term demand for software engineers is far in excess of supply.

Software project control panel. The Initiative needed to provide more than 400 DoD project managers with a useful means of obtaining visibility in their large-scale software projects. The result is the software project control



panel, shown in Figure 2. Similar to an aircraft pilot's control panel that provides critical information regarding the aircraft's status, the software project control panel displays current project status. This includes leading indicators of slowdowns and breakdowns as well as potential problems in risk and quality.

WHERE DO WE GO FROM HERE?

First, we must change our mindset. Software organizations must move from a checklist mentality and a quest for the silver bullet to a process where proven practices are the foundation for management. The focus must turn to the bottom line: productivity, quality, timeliness, and user satisfaction. Process improvement efforts must directly address and accomplish these fundamentals.

Second, at least with regard to government software acquisition, government contracting incentives must be

overhauled, for they currently penalize effective performance and reward inefficiency. Potential changes are being explored under the DoD Software Acquisition Best Practices Initiative.

Third, many project managers would greatly benefit from becoming more familiar with key management techniques and issues. This poses a very big educational and training challenge, and the Software Program Managers Network is working to identify and formulate the knowledge and skills required to implement software best practices.

The practices I've described have been drawn from the crucible of successful real-world projects. There's no reason why they can't be effectively implemented by any organization developing or maintaining large-scale software. They are intended as a set of management practices to prevent high inherent complexity from growing into

unmanageable chaos. The nine practices I've described are not exotic or glamorous. Ironically, the techniques that really seem to work in managing high-tech software are low-tech.

Organizations that don't use these practices will likely find it increasingly difficult to compete in the global marketplace — and will likely fulfill Ed Yourdon's prophecy that those with low software productivity and quality will suffer the same fate as the dinosaur.¹³ ♦

I invite software practitioners and managers to join the Software Program Managers Network membership. The products referred to in this article are free from the Network's home page (<http://spmnet.com>), or you can send e-mail to member@spmnet.com; (703) 521-5231 voice; (703) 549-9583 fax.

REFERENCES

1. Dept. of the Air Force Software Tech. Support Center, *Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapons Systems; Command and Control Systems; Management Information Systems; Vol. 1*, Washington D.C., Feb., 1995.
2. G.H. Porter, *Memorandum for Service Acquisition Executives, Program Executive Officers, Program Managers*, Office of the Secretary of Defense, Washington, D.C., Apr. 1994.
3. C. Jones, *Applied Software Measurement*, Second Edition, McGraw-Hill, New York, 1991.
4. *Defense Report of the Defense Science Board Task Force on Military Software*, Office of the Under Secretary of Defense for Acquisition and Technology, Washington, D.C., 1987.
5. N. Longuemare and E. Paige Jr., "DoD Policy on Software Acquisition Best Practices," Office of the Secretary of Defense Joint Memorandum, Washington, D.C., July 8, 1994.
6. R. S. Pressman, *Software Engineering, A Practitioner's Approach*, McGraw-Hill, New York, 1992.
7. B. Boehm, *IEEE Tutorial on Software Risk Management*, IEEE CS Press, Los Alamitos, Calif., 1989.
8. W. S. Humphrey, *Managing the Software Process*, Addison-Wesley, Reading, Mass., 1989.
9. M. C. Paulk et al., "Capability Maturity Model, Version 1.1." *IEEE Software*, July, 1993, pp. 18-27.
10. M.E. Fagan, "Advances in Software Inspections," *IEEE Trans. Software Eng.*, July 1986, pp. 744-751.
11. W. S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
12. L.M. Marable, "The Fifty Hottest Jobs in America," *Money*, Mar. 1995, pp. 114-117.
13. E. Yourdon, *Decline and Fall of the American Programmer*, Prentice-Hall, Englewood Cliffs, N.J., 1992.



Norm Brown is the founder and executive director of the Software Program Managers Network, a member (ex officio) of the DoD Software Management Review Board, and executive director of the DoD

Software Acquisition Best Practices Initiative. As such, he is responsible for identifying, articulating, and effectively implementing software best practices throughout the Department of Defense. Brown provides counsel and assistance to program managers and their staff in addressing and resolving critical software issues and has developed focus teams to provide specialized support.

Brown received a BS in electrical engineering from Pratt Institute, an MS in electrical engineering from Rutgers University, and a JD from American University. He is a member of IEEE and ACM