CONFIGURING FOR SUCCESS

By Mike Hill

Suppose that your team just rebuilt the on-line order entry program to fix a performance problem, only to find that it runs more slowly and inserts duplicate entries into your customer database? You may have had related problems before that. For instance, on the day before releasing it for production, the application would not compile because a programmer had changed a line of code in an apparently unrelated module. Perhaps you agreed last month to provide social security number references to the customer table, only to realize later that you would have to authorize overtime in order to make the changes on schedule. Maybe a month earlier, the team lost the development code library while preparing a demonstration prototype, and had to rebuild the product baseline from earlier source code versions.

Such problems are not uncommon for organizations moving from mainframe to client/server and object-oriented technologies. Software development is undergoing a general crisis where more than **one-half** of all large projects overrun both schedule and cost, **88 percent** requires substantial redesign, and a whopping **25 percent** experiences outright cancellation.[1] Often, the faults occur in change management and quality control practices. Why does this happen, when we staff only the most qualified professionals, carefully specify requirements and diligently address user expectations?

Dave Barry might interject at this point: "I know you probably have a lot on your mind already, but you should be aware that 90 percent of the universe in missing".[2] The problem is in the process, or lack of it; software projects fail because designers do not employ effective engineering practices to manage complexity.

The IEEE formally defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software".[3] However, such basic engineering concepts seem foreign to many software enterprises.

There may be several reasons for this. Academic Computer Science programs lag far behind real-world practice, with the result that many software professionals lack formal training in engineering principles.[1,4] Programmers, analysts and managers possess different background experiences that impede development of a common understanding of key concepts.[4] Misunderstanding unfamiliar terminology and technical detail may lead an unseasoned manager to believe that, far from helping, using engineering methodology will only delay the project.

Yet, a formal engineering process *is essential* to managing the size and complexity of projects that involve many contributors. It organizes and accounts for all the many parts of the system, as well as all the relationships among those parts. It allows any team member or manager to view the

whole system, to track items necessary for completion and to coordinate delivery of error-free software on time and on budget.

Software Configuration Management

Software Configuration Management (SCM) is an established software engineering discipline that provides techniques for meeting the critical project objectives of schedule, budget and quality. A *software configuration* is the arrangement of components and their interconnections that make up one version of a computer software system. A *configuration item* (CI) is a software component such as a source code file, object module, build script, database procedure, user interface image, executable program, documentation or any combination of these.

SCM is a structured process that transforms a system design into the functional and physical characteristics required for release of the software configuration to production. In addition to coordinating and tracking system construction, it identifies and controls changes made by programming teams.[5] Managing change also extends to system maintenance and enhancement following production release. Thus, SCM covers three phases in the life cycle common to all software projects: development, delivery and maintenance.

Basic SCM Process Model

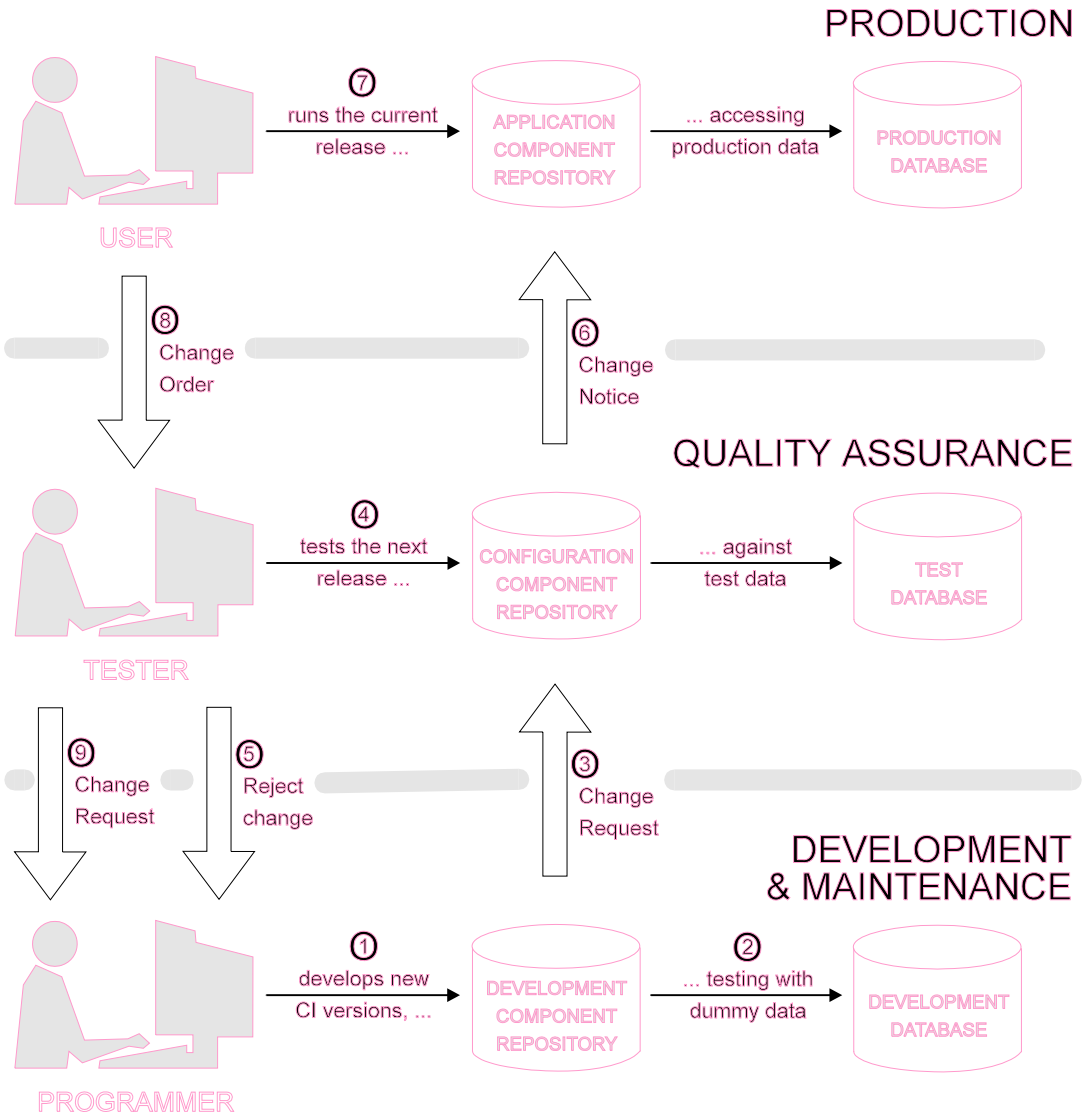
As shown in the diagram, the basic SCM process model consists of three layers, each containing its own database and software component repository: Production, Quality Assurance and Development and Maintenance. The Production layer contains the system products of development and testing, as delivered for current operational use. Quality Assurance holds the configuration baseline that will comprise the next system release. The system's initial development and ongoing maintenance occur at the Development and Maintenance level.

Configuration items move from one SCM layer to another based on the success of testing. A Software Configuration Administrator (SCA) coordinates the process via formal *Change Orders*, *Change Requests* and *Change Notices*. This is similar to Engineering Change Notice procedures used in other engineering disciplines. The SCM process consists of the following steps, corresponding to the numbers in the diagram:

1. A programmer checks out an existing CI from the Development Component Repository and modifies it per requirement or specification. Check-out prevents other programmers from altering the CI.
2. The programmer completes unit testing and checks the CI into the repository. It receives a new version number and becomes available for use and modification by other programmers. Item check-in also may trigger compilation, database entity creation or other appropriate processing.

3. The programmer then submits a Change Request for promoting the CI to Quality Assurance. The SCA rebuilds the configuration, incorporating the new CIs.
4. Testers use exhaustive Software Quality Assurance measures to prove correctness and reliability of the configuration. These measures normally include system, acceptance and performance testing.
5. If a CI fails testing, the Tester rejects it. The SCA restores the previous CI version and directs the original Change Request back to Development and Maintenance with a problem description.
6. The SCA issues a Change Notice that incorporates the CIs passing Quality Assurance. The SCA then assigns the configuration a new release number and moves it to Production.
7. Production users note difficulties and desired enhancements, which the SCA maintains in problem logs.
8. The Production users submit Change Orders for the desired changes. The SCA routes these to Quality Assurance along with the problem logs.
9. Testers verify errors and assess enhancements, and then prepare new Change Requests to address specific Change Order requirements. The SCA routes the new Change Requests to the appropriate programmers, and the cycle continues as before.

SCM Process Model



This model has proved effective in a variety of development and reengineering projects such as petroleum production and exploration, semiconductor manufacturing and market show registration. It has supported iterative development processes that included frequent prototypes and deliverables, as well as traditional "waterfall" development life cycles. The most successful implementations employed simple but robust check-out/check-in version control, electronic change control forms and distributed configuration control. The last factor was most important: Programmers managed their own Development and Maintenance repositories and databases, bypassing clumsy and slow centralized configuration management procedures.

Administrative Roles

Project management, software quality assurance, database administration and configuration administration activities all participate in the SCM operation's success. The Project Manager (PM) ensures that all team members understand and adhere to the SCM process. The PM also negotiates Change Orders, prioritizes changes and approves all Change Requests. Software Quality Assurance (SQA) specialists integrate SCM into the SQA Plan, develop test plans and procedures, and oversee validation and verification procedures. A Database Administrator (DBA) implements the SCM repositories and databases, maintains the changing data model and enforces access security. The DBA also rebuilds the Production database for each new release.

The pivotal SCA role (alternately referred to as the Source Code Administrator, Software Librarian or Configuration Control Board) applies SCM techniques transparently to the development and production efforts and maintains product integrity. This requires someone who knows the end product and its specifications, who understands the engineering process and its goals, and who can make it work in a team environment using automated software development tools. In any other type of engineering project, the person in this role would hold the title of "Chief Engineer". Mobil's MEPSI GUIDE and Texas Instruments' MMST projects provide good examples of software configuration administration.

Process and Tools

Any project involving more than one person needs SCM automation. The larger and more distributed the project, the more important it is to automate the process. Multi-user databases, source code control systems, version control software, and e-mail significantly aid a project's success.[5,8,9]

ISO guidelines and IEEE standards describe the basic SCM requirements. These include: SCM planning, component identification, change control, status accounting and reporting, audits and reviews, interface control, scheduling and resource management.[10,11,12] Other sources add version control, configuration building, software library management, software migration and distribution, standards checking, process workflow management, access control, problem reporting and even help desk interaction.[7,13] Ideally, project automation should support all of these needs, tailored to specific life cycles.

Tools exist to automate integrated software synthesis and management across virtually all system life cycle phases. However, there appear to be no "silver bullets" to address all configuration management needs. The available tools generally lack the capabilities necessary to support industrial-strength software projects. Also, effective tool utilization often requires extensive manual integration and interpersonal communication. Other commonly noted deficiencies involve the areas of defect tracking, distributed configuration control, concurrent development, tool incompatibilities, third party and off-the-shelf component integration, binary file

maintenance, documentation and overall process integration.[7,9,13] *Paradoxically, the most severe inadequacies appear to be in change management and quality control.*[4]

A rigorous software engineering discipline will overcome deficiencies in automation tools, and therefore should drive their selection. Acquire the tools after defining the project life cycle, establishing the SCM plan and determining how to control change and quality. The tools should support the process; not replace it.

Get a Grip

A well-designed SCM plan imparts a consistent product engineering discipline that allows project team members to work together productively throughout the project life cycle. Programmers work from a systems perspective rather than creating isolated applications and reports. The structured environment prevents mixing production, test and development code. Even an informal software change control mechanism will yield immediate benefits.

SCM is a vital ingredient in any software project. It occurs as the common denominator in several standards and guidelines, and forms the core of most software engineering activities.[6,10,11,12] Also, the Software Engineering Institute identifies it as a key process area required for reaching Level 2 of the Capability Maturity Model, where a software organization achieves process stability and repeatable management control.[14]

By attaining a level of comprehensive process and quality management, you minimize the risks inherent in introducing advanced technologies into your organization. Get a grip on your project's critical success factors by including SCM in the software engineering process.

Notes

[1]Gibbs, W. Wayt. "Software's Chronic Crisis," *Scientific American*, Vol. 271, No. 3, September 1994, pp. 86-95.

[2]Barry, Dave. "The Universe Has Gone to the Dogs," Tribune Media Services. The Dallas Morning News, January 15, 1995, p. 10F.

[3]The Institute of Electrical and Electronics Engineers, Inc. Glossary of Software Engineering Terminology. IEEE Std 610.12-1990. New York, NY, 1990.

[4]State of Practice Working Group, IEEE Computer Society Task force on ECBS. "Systems Engineering of Computer-Based Systems," *Computer*, Vol. 26, No. 11, November 1993, pp. 54-65.

[5]Bersoff, Edward H., and Davis, Alan M. "Impacts of Life Cycle Models on Software Configuration Management," *Communications of the ACM*, Vol. 34, No. 8, August 1991, pp. 104-118.

[6]The Institute of Electrical and Electronics Engineers, Inc. Standard for Software Quality Assurance Plans. IEEE Std 730.1-1989. New York, NY, 1989.

[7]Buckley, Fletcher J. "Implementing a Software Configuration Management Environment," Computer, Vol. 27. No. 2, February 1994, pp. 56-61.

[8]Tristram, Claire. "Do You Really Need ... Version Control Software," *Open Computing*, Vol. 11, No. 12, December 1994, pp. 75-78.

[9]Nix, Kevin. "Implementing Process Configuration Management," *Computer Design*, Vol. 34, No. 1, January 1995, pp. 124-126.

[10]International Organization for Standardization. Quality Management and Quality Assurance Standards -- Part 3: Guidelines For The Application Of ISO 9001 to the Development, Supply and Maintenance of Software. ISO 9000-3:1991(E). Geneva, Switzerland, 1991.

[11]The Institute of Electrical and Electronics Engineers, Inc. Guide to Software Configuration Management. IEEE Std 1042-1987. New York, NY, 1987.

[12]The Institute of Electrical and Electronics Engineers, Inc. IEEE Standard for Software Configuration Management Plans. IEEE Std 828-1990. New York, NY, 1990.

[13]Fuggetta, Alfonso. "A Classification of CASE Technology," *Computer*, Vol. 26, No. 12, December 1993, pp. 25-38.

[14]Saiedian, Hossein, and Kuzara, Richard. "SEI Capability Maturity Model's Impact on Contractors," *Computer*, Vol. 28, No. 1, January, 1995, pp. 16-26.