*Feature Article:*

# Making the Case for Software Management

***By Paul Conte***

*This article is excerpted with permission from the* ***Making the Case for Software Management*** *whitepaper by Paul Conte, available at no charge from SoftLanding Systems at www.softlanding.com. The full whitepaper provides a step-by-step guide, including a list of ten "checkpoints" to help you "make the case" in your own organization. Paul is also author of* the ***Software Development Survival Guide – Five Steps from Chaos to Control*** whitepaper available from SoftLanding Systems.

If you're just beginning to explore ways to improve your IT organization's software management practices, you may want to read *Software Development Survival Guide – Five Steps from Chaos to Control*, which explains the importance of software management and describes the following five steps to get started:

- Step 1: Make the Case
- Step 2: Adopt a Change Management Process
- Step 3: Adopt a Quality Assurance Process
- Step 4: Adopt an Incremental and Iterative Development Process
- Step 5: Continually Assess and Refine Processes

As that whitepaper advises, you should begin by making a convincing case to your own management and staff before you attempt any big changes. If you're ready to take that step, this article will help you chart your campaign.

## Know where you are

Every IT organization practices some form of software management, whether ad hoc and mostly manual, or well-defined and highly-automated. So the case you'll be making isn't really to have your IT organization *adopt* software management. Rather, you'll be making the case to *improve* your organization's software management practices in some concrete way. As a first step, be sure you have a concise, up-to-date description of your organization's current software management practices, as described in the *Software Development Survival Guide*.

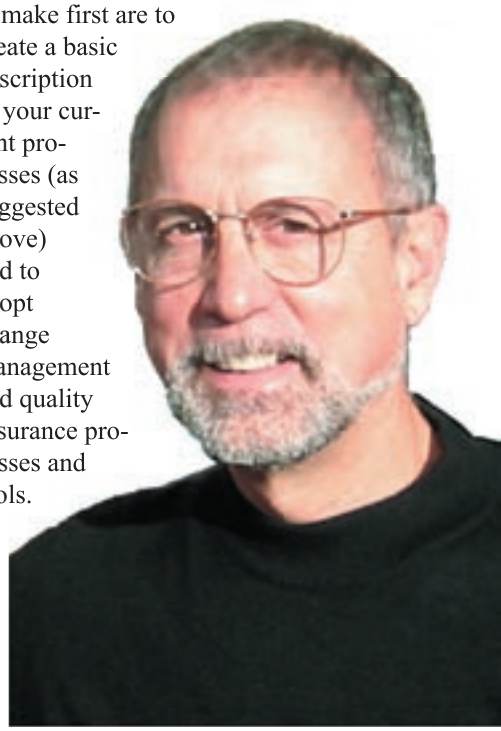
## Identify key problems that need addressing

After you establish what your current practices are, assess where you currently have serious problems in consistency, function, quality, timeliness, cost, or service. The following list of questions can help:

- Do end users complain that:
- User error reports and feature requests aren't prioritized and scheduled in a coherent way?
- They frequently can't determine the current status of an error report or feature request?
- Can some application artifacts be used or modified by people who shouldn't have access?
- Is it sometimes difficult to locate a particular application artifact and determine its modification status?
- Is it difficult to rebuild current and/or previous production releases of an application from scratch (i.e., compile and deploy all the necessary artifacts)?
- Do new errors get introduced or existing errors go unfixed when you make changes to application software?
- Do problems emerge during deployment that went undetected earlier?
- Are projects commonly initiated without allocating specific levels of staff time and other resources?
- Are new or modified applications sometimes deployed in production without thorough testing?
- Do developers regularly get calls outside of work to handle "emergency" application failures?
- Do software projects commonly exceed planned schedules and/or budgets?

## Know where you're headed

If your organization has very few defined software management practices, the most likely improvements you need to make first are to create a basic description of your current processes (as suggested above) and to adopt change management and quality assurance processes and tools.



If you've already incorporated CM and QA into your organization, I suggest you adopt an incremental and iterative software development process as your next step.

A bit of advice: even if you're an enthusiastic advocate of software management "best practices," don't try to make the case for all potential improvements at once. Pick those items you feel confident you can accomplish and that will have a clear benefit to the enterprise. Then build on your successes to make further improvements.

## Identify the specific support you want

Once you've decided which improvements to tackle, create a bulleted list of the *specific actions* you want management and staff to support. Note the types of costs or potential negative impacts (money, time, staff objections) associated with the item. Later, you'll have to quantify these, along with identifying benefits.

As you proceed, keep the list concise and current and use it to maintain focus in your discussions with management and staff. A clear statement of what you want from management will let everyone involved know what's really at stake and will keep them from getting sidetracked by concerns that aren't relevant.

## Know your audience

Most likely, your decision to make a case for CM is (or will be) based on the fairly strong belief that CM's benefits warrant the cost and effort. However, managers and other staff may have a very different set of interests and concerns in their jobs than you do in yours. You need to build your case to address *their* mindsets, not your own.

***Corporate-level managers*** are looking for ways to improve the company's profits, or delivery of services, in the case of government and other nonprofit organizations.

***IT managers*** obviously are concerned the IT department deliver capabilities requested (or demanded) by corporate-level managers. But, in my experience, the dominant focus of most IT managers is minimizing various types of risk, including lengthy system outages, major application failures, or exploding project schedules and budgets.

***Software developers*** like to cut code; they don't like to be recognized as the source of an application deficiency or failure; and, for the most part, they hate "maintenance" programming. Your case to them should paint a believable picture of how software management improvements will increase the time they spend doing "fun" work and reduce their interruptions and frustrations.

## Create a skeleton benefits summary

It's time to sketch out the case you'll present to win support for your proposed changes. One way to get started is to create a skeleton "Benefits Summary" document, as in the following example:

## Benefits Summary

Establishing a basic Change Management process and using a tool to automate CM practices will:

- Improve our sales revenue by increasing availability of applications that support in-house sales representatives and Web-based direct sales.
- Strengthen our market position by reducing the time-to-market of strategic applications.
- Reduce the risk of application failure, thereby reducing consequent revenue loss and the expense of software fixes.
- Reduce the cost of delivering new or improved application features by increasing the percentage of time that software development staff spend implementing new capabilities versus correcting errors.
- Reduce the risk of application development projects exceeding budget or schedule, thereby avoiding unanticipated costs and delays in executing corporate strategies.
- Improve IT staff performance and morale, thereby improving the IT department's ability to deliver new strategic software projects.

## Develop supporting arguments

Next you need to develop compelling arguments that support the benefits you believe your proposed changes will produce. Here are several types of arguments you should consider.

***"Reduce risk" argument***

Most organizations – and their managers – want to minimize risk, or at least take risks only when the potential return is commensurately high. In a poorly managed IT environment, however, you typically find many *avoidable* risks with a high potential cost and low potential gain. For example, when an organization doesn't have an application deployment (or "build") process that can be used to reliably produce a complete, correct, and properly secured production release, there's a high risk of the following:

- Excessive personnel costs to produce a properly working release
- Delays in the initial availability of a new release
- Functional defects that limit the usefulness of new (or previously available) features
- Failures that make the application unavailable
- Unauthorized access by "hackers" (or internal users) that corrupts or steals vital data.

A "high risk" deployment process may result in substantial costs to the organization, including lost sales revenue, reduced customer satisfaction, damaged or stolen corporate information, and substantial staff time for recovery. But what is the potential gain for an organization that assumes the risk of an unreliable deployment process? Practically nothing!

In some situations, you may be able to quantify some of the risks that exist when you lack CM or other basic software management processes. But, in many cases, you can make an effective argument simply by asking management to answer questions such as the following:

- Is it acceptable for a new application release to be available up to two days later than scheduled due solely to deployment problems?
- Is it acceptable for a new application release to have twenty percent

higher reported problems in the first week of use due to "build" problems?
- Is it acceptable to have sensitive data exposed to unauthorized personnel due to "build" problems?

You should be able to develop similarly effective questions from your list of the major benefits you expect from proposed software management improvements.

***"Accountability" argument***

Managers of well-run organizations generally are concerned with accountability and auditability. They want to know which people and practices were responsible for significant outcomes or events – good or bad.

The Sarbanes-Oxley Act of 2002 codifies this principle by mandating public companies doing business in the U.S. have a comprehensive accounting framework that ensures all financial performance results are backed up by substantiating data that's readily available for follow-up audits. Because a company's financial performance and reporting is intertwined with software applications, Sarbanes-Oxley reinforces the need for software management practices that provide accountability and verifiability.

A glaring problem for many organizations is the lack of adequate controls and audit trails over application modifications, testing, and deployment. Using the tactic introduced in the previous section, you might pose questions such as the following to management:

- Is it acceptable for a developer to make unauthorized changes to an application?
- Is it acceptable to have no code inspection, testing, or other process to verify that application changes produce the intended results?
- Is it acceptable to have no audit trail of application modifications?

The prospect of acknowledging any of these possibilities to an auditor should bring chills to any corporate-level manager.

***"Reallocate unproductive staff time" argument***

If your IT department lacks effective software management, it's likely that significant staff time is being spent fielding end users' error reports and complaints, correcting programming errors, manually cobbling together new application releases, and a raft of other activities that don't produce any real benefit to the organization.

You'll bolster your case substantially if you can convince management your proposed improvements will allow reallocation of significant staff time from nonproductive tasks to productive tasks, such as application enhancements that produce revenue, reduce costs, or improve service.


Solutions Beyond Limits...
eServer has IT all!

TUG™

April 20 - 21, 2004
Sheraton Parkway Hotel

The Toronto Users Group for Midrange Systems (TUG) once again presents the annual Technical Education Conference and Showcase (TEC). We will feature the latest hot topics in the eServer realm and the topmost expert speakers. For more information contact the TUG Office at 905-607-2546 or e-mail: admin@tug.ca.

TUG TEC2004

***"Cost of unavailable application(s)" argument***

If your company has applications whose availability is critical to revenue, you may be able to estimate a credible cost per hour when such applications are unavailable. In most cases, developing a figure that won't be questioned by corporate management requires the help (and subsequent backing) of senior staff or management in the affected operational department. For example, the sales department staff may be able to estimate the lost (and unrecoverable) sales revenue from a Web-based retail store being unavailable during different time intervals throughout the week.

***"Cost of delayed deployment" argument***

If corporate-level managers have identified a potential application they believe will increase revenue or market share, reduce costs, or otherwise produce a quantifiable benefit to the organization, you may be able to estimate a credible cost for delays in deploying the application. For example, if a new Web-based retail store application is forecast to increase sales revenue by at least $1,000,000 in the first year, you might use a simple calculation showing that each month of delay results in a loss of over $80,000 of potential revenue. And, of course, there are direct IT personnel costs and other costs for each extra day of project duration that you can probably estimate fairly well.

***"Cost of errors" argument***

The cost of errors has surfaced in several of the previous arguments, but this issue deserves some consideration in its own right. If you have any reasonable basis for calculating the approximate IT staff time spent correcting errors (including diagnosis, correction, redeployment, and so on), and you can make a sound case that some significant percentage of these errors can be prevented by a proposed software management improvement, you should include this as one of your lead arguments. Two of the software management areas to which you can usually apply this argument are application deployment and regression testing.

## Develop reasonable estimates of the cost and effort to make the improvements

As one of your first steps, you created a bulleted list of the *specific actions* you want management and staff to support and noted the type of costs or potential negative impacts (money, time, staff objections) associated with each action. As part of your complete case, you need to flesh these impacts out.

Because at this stage you may not know the exact cost of products or even some of the specific practices you may adopt, I recommend you use ranges of values and conditional impacts, as appropriate. For example, a quick survey of available CM products for a company your size may indicate the likely costs to be between $8,000 and $40,000 for the initial license and between $1,600 and $8,000 annually for support. Software management tool vendors and their customers are good sources of data to support your estimates for training and impacts on personnel time.

## Overcoming common objections

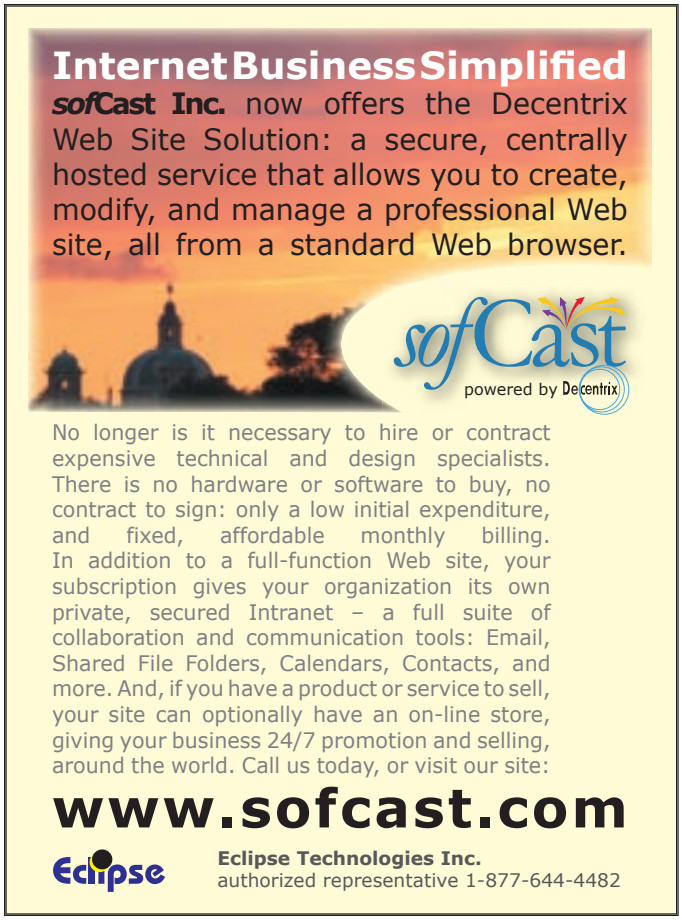
As a final step before you roll out your proposal and the supporting arguments, try to anticipate and prepare for likely objections. One way to do this is to ask people who you think are generally supportive of your proposal to play "Devil's Advocate." It may also help to anticipate your response to the common objections I list below.

***"Why do we need to do this now?"***

If you've already identified concrete revenue increases, cost reductions, service improvements, or other concrete benefits, then realizing these benefits sooner increases the net payoff. If you're about to embark on a major new application, improving software management beforehand can reduce risks. Finally, if your company is concerned about new requirements for accountability and auditability, such as those mandated by the Sarbanes-Oxley Act, you may be facing compliance deadlines that need prompt action.

***"If this is such a good idea, why haven't we done it already?"***

The honest answer to this question may be: "We should have been doing it, but we weren't." A more diplomatic approach is to present some of the same responses you would to the previous question. For example, the adoption of J2EE or .NET – which weren't part of your previous IT environment – may provide a satisfactory external justification for the timing of your proposal.

***"Our IT department is too small to need this."***

Is the whole company also too small to worry about profitability, market share, risk, legal requirements for accountability, and so forth? To meet this type of objection, you need to make a strong connection with the corporate-level management and their concerns. You also need to pick a limited set of improvements that won't be onerous for a small staff to adopt and which will have direct, short-term payoffs. Some good candidates are: lightweight source control, automated deployment, testing and debugging tools, and iterative and incremental development.

***"This will just increase overhead and won't save any time."***

You should take this as simply a "show me" challenge, which implies nothing more than that the cost of your proposed changes will exceed the benefits. You must effectively address this issue in any case, whether or not the challenger expresses the concern in such a "hostile" manner.

***"We're too busy to make these improvements right now."***

This is the "granddaddy" of all objections to improving software management because the *future* benefit of software management improvements runs squarely into the *immediate* reality that many such improvements will require additional time or money before the benefits result. You need to face this fact in your proposal and ask support for actions such as increased budget, additional personnel, temporary project deadline adjustments, or other steps to assure your IT organization *can* successfully make the proposed changes.

## Conducting your campaign

As you conduct your campaign, look for ways to adapt your initial proposal as you discover which elements seem to be well received and which don't go over well. If you were faced with questions that can be answered adequately by further homework, then putting more effort into bolstering your arguments in support of a particular action is worthwhile. On the other hand, it may be better to prune some actions from your proposal and revisit them after you have some success under your belt. The main point is to avoid getting "stuck" on the exact composition of your proposal's initial version. Think of it as the first step on a path of continuous improvement leading to a final proposal that will have the wholehearted support of your organization.

## Delivering on your promise

Finally, and briefly, because this article is about making the case, not implementation, be sure you continue your effort to ensure subsequent software management improvements you make result in recognizable success. As an example, don't just buy a CM tool and implement an automated deployment process. If you haven't already got this data, take time to record the effort required by your current manual approach and then measure the improvements after you implement an automated process. Write up brief descriptions of observable improvements in other areas, as well, such as reduction in errors from better testing practices. The best way to start your next campaign for further software management improvements is to make sure your audience is well aware of your previous successes. TUG


© The 5th Wave, www.the5thwave.com

***Paul Conte** is president of Picante Software, a consulting firm in Eugene, Oregon, and a senior technical editor for e-Pro Magazine and iSeries NEWS. Paul has published numerous articles on software development practices and is a widely-recognized authority on database technology. He is author or co-author of five books, including SQL/400 Developer's Guide and SQL Server 2000 Developer's Guide. Paul has a BA in Psychology from Georgia State University and an MS in Computer Science from University of Oregon.*