How long can you wait for CMMI Compliance?

Manage your software development in guaranteed compliance with the CMMI® NOW!

processMax includes all the necessary policies, procedures, guidelines, criteria, templates, and forms in role-based, step-by-step instructions, ready for use—everything you need for CMMI compliance. Integrated with robust document management and workflow, processMax is the intranet web-based solution for effective and efficient management of your software development projects. processMax also includes a complete measurement and reporting capability. Measurement collection and report generation are fully automated—giving managers real-time, fact-based insight into all aspects of project progress. processMax is the proven solution—as verified by more than 50 successful independent appraisals.

processMax
THE PROCESS SOLUTION

www.pragmasystems.com

pragma Systems Corporation, 1810 Samuel Morse Reston, VA 20190
877.838.PMAX  E-mail: info@pragmasystems.com
processMax is a registered trademark of pragma SYSTEMS Corporation.
CMMI is a registered mark of Carnegie Mellon University. Copyright © 2005 pragma SYSTEMS Corporation.
The theme for this issue of Software Tech News (STN) is measurement, especially as it relates to integration and software process maturity. Integration is capturing more and more of our focus and our effort, regardless of what we call it. At the project level, within the boundaries of a single software system, we often focus more on integrating existing systems and COTS than developing a system from scratch. At the organizational level and across organizations there is the need to integrate existing software systems, as components of an enterprise, in order to enable business intelligence (sometimes called “knowledge management”). While the commercial community uses the terminology of “enterprise”, in the military domain, integration is often addressed from the perspective of the System-of-Systems concept (or a federation), enabled by “Net-Centricity”. There is also the trend to integrate management functions, as evidenced by the growth of Project Management Offices (PMOs) to oversee the projects and integrate the management (and measurement) of multiple projects. This growing focus on integration, and its counterparts, may evolve into a discipline that is distinct from the Software Engineering discipline, as we know it today. It also creates new demands for measurement. While the measurement methodologies of the past decade are still relevant, we have to find better ways to plan for and specifically address integration related issues.

In their article titled “The Measurement Challenge of High Maturity”, Domzalski and Card describe BAE’s journey to the Capability Maturity Model Integration (CMMI) Level 5 and how they evolved their measurement program to support that journey. Noteworthy is the fact that they actually discuss what failed as well as what worked.

Note that CMMI itself has evolved from several separate maturity models of the past decade (Capability Maturity Model for Software (SW-CMM), Systems Engineering Capability Model (SECM), and the Integrated Product Development Capability Maturity Model (IPD-CMM), and others).

CMMI integrates a set of processes, which combine Systems Engineering, Software Engineering, Project Management, and Organizational Process Improvement along with many support functions and was intended for use by organizations in pursuit of enterprise-wide process improvement.

Their article focuses on actionable measurement, that is, using measurement data with confidence, to predict, and to communicate variances from the predictions, in a way that alerts decision makers to potential problems in a timely manner so that they can act and affect the outcome. This capability is possible because of several years of building a measurement data repository.

Domzalski comments that much of the improvement activity (predictive capability) of higher maturity organizations is based on historical measurements previously recorded for “possible future use”. Yet he also states that lower maturity continues on page 2
organizations tend to collect too much data and use too little of it. This is reiterated in Hawald’s article as well. Most measurement paradigms tell us to collect only what we need, because collecting and storing the data is time consuming and costly. How then, does a fledgling organization make this determination about what constitutes too much data? How do they determine the potential future value of their data? How far out should an organization be looking when planning its measurement program? Perhaps this is a well-kept secret of those mature organizations — knowing what to collect and when to start.

Domzalski also talks about “noise” in the measurement data collected. Recognizing that data noise exists is more important perhaps than trying to eliminate it. For example, in order to collect effort data, does one require staff to separately report exact hours actually worked on a particular task, or, is it better to automate the collection by using time card data? The noise relates to staff time not actually working (coffee breaks, discussions that go off target, etc). The noise level may differ among groups, or even individuals, but the overriding goal, for measurement purposes, is to achieve consistency over time and across projects. Accepting the noise may be a tradeoff for the benefits of automated and consistent data collection.

In the 2nd article titled “Estimating System-of-Systems (SoS) Development Effort”, Jo Ann Lane asserts that existing cost models do not handle integration of SoS well. Her article then focuses on recent work in developing a cost model that specifically addresses how to budget for SoS integration activities including the up-front efforts associated with SoS abstraction, architecting, source selection and systems acquisition, as well as the effort associated with integration, test, and change management. This new model, called the Constructive SoS Integration Model (COSOSIMO), is an addition to the COCOMO suite of estimation tools. It is also distinct from the Center for Software Engineering (CSE) System Engineering Cost Model (COSYSMO), which is used to estimate the system engineering effort at the single system level.

In the 3rd article, titled “Measurements: Managing a Large Complex Modernization Effort — While Protecting Your Project From the ‘Katrina Factor’”, Steven Hawald asserts that Project Management Offices (PMOs) need to adopt a strategic view of their measurement program that addresses how measurement will be needed in the future. His analogy to the Katrina tragedy hints at the serious impact that a lack of appropriate measurement planning can have on an organization. He indicates that few organizations are investing enough time or money into developing a comprehensive project performance measurement program and he discusses three key points for reinforcing your data and measurement “levees”.

The last article, by Robin Ying, titled “Building Systems Using Software Components”, discusses metrics that are critical for building trustworthy systems, namely, reliability, security, and safety. He cautions that we cannot simplistically assume (based on mathematics) that if all the components we select are reliable, then the resulting system will be reliable. The act of integration introduces errors. As the number of components in a system increase, the number of integration errors increases as well. He sites careless integration and careless reuse as responsible for several disasters (Mars Climate Orbiter disaster), since a single integration error can make the whole system unreliable, even if all of its components are fully reliable. He asserts that the reliability of a software system should be judged holistically, emphasizing the functional relation between the components and the system as a whole. He concludes that, among other things, developing trustworthy software systems requires continuous and undivided attention to quality.

About the Author
Ellen Walker, a DACS Analyst, is currently developing a series of publications on software “best practices” as part of the DACS Gold Practice Initiative. She has spent the past 20 years as a software developer in various roles spanning the entire software life cycle including project management of multiple business process re-engineering efforts within the DoD community. She is also experienced with assessment initiatives such as the Capability Maturity Model for Software (CMM-SW) and the quality management practices of the New York State Empire State Advantage program. Ellen has an MS in Management Science (State University of New York (SUNY) at Binghamton), and bachelor degrees in both Computer Science (SUNY – Utica/Rome) and Mathematics (LeMoyne College).

Ph: 315-334-4936
Email: ellen.walker@itt.com
The DACS Gold Practice Initiative:

- Promotes effective selection/use of software acquisition & development practices
- Defines essential activities/benefits of each practice
- Considers the environment in which each practice is used
- Addresses the timeliness of practice benefits
- Recognizes interrelationships between practices that influence success or failure
- Contains quantitative and qualitative information
- Includes discussion forums, case studies and survey opportunities
- A continually evolving resource for the DoD, Government, Industry and Academia
- Free to use/free to join

Current Gold Practices:

- Acquisition Process Improvement
- Architecture-First Approach
- Assess Reuse Risks and Costs
- Binary Quality Gates at the Inch-Pebble Level
- Commercial Specifications and Standards/Open Systems
- Ensure Interoperability
- Formal Inspections
- Formal Risk Management
- Goal-Question-Metric Approach
- Integrated Product and Process Development
- Model-Based Testing
- Plan for Technology Insertion
- Requirements Management
- Requirements Trade-Off/Negotiations
- Statistical Process Control
- Track Earned Value

Learn More About the DACS Gold Practice Initiative:
http://www.goldpractices.com

Become a Registered Member!
http://www.goldpractices.com/registration/register_step1.php

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400
http://iac.dtic.mil/dacs
The Measurement Challenge of High Maturity
By Kevin Domzalski, BAE Systems
With assistance from David Card, Q-Labs

Introduction
CMMI Level 5 is far too often erroneously thought of as an idyllic state of perfection. Much of the hard work it takes to get there, and stay there, and the many experiences gained along the way may be forgotten in the afterglow of success. Many of the most difficult obstacles on the road to high maturity are related to measurement and analysis.

The purposes of this article are to
• Discuss some of the measurement problems and difficulties that are encountered on the road to CMMI Level 5
• Describe some of the measurement techniques that proved especially useful to us
• Outline some of our plans for going forward, recognizing that Level 5 is not the destination, only the end of the beginning

Let's start with reviewing how we got to where we are today.

Our History with CMM/CMMI
In 1989, BAE Systems National Security Solutions, then a division of General Dynamics Corp., took our first steps on the path to high maturity practices by undertaking an assessment using the Carnegie Mellon University Software Engineering Institute's (SEI's) Software Capability Maturity Model (CMM®) and rating ourselves at Level-1. Over the next six years, the organization advanced at a reasonable pace, achieving Software CMM Level-2 in 1992 and Level-3 in 1995. The organization then expanded its scope beyond software engineering and, in 1997, achieved Level-2 in both the Systems Engineering (SE) and People CMMs. In 2000, Systems Engineering was assessed at Level-3 (SE-CMM).

Also in 2000, our organization confronted its first major process improvement setback when it attempted and failed to be assessed at Software CMM Level-4. (You can get a good feeling for the true maturity of an organization by watching them deal with failure.) The 2000 assessment results were, to say the least, eye opening.

The general improvement opportunities that were identified focused on the following:
• An expansive (and “too complex”) process documentation set
• A general lack of stability and control of our measurements, at both the project and organizational levels, and
• A weak understanding of the implications of process and technology change in quantitative terms.

Figure 1. CMM/CMMI Appraisal History
Measurement Challenge  Continued from page 4.

During 2001, our company hand-selected over two dozen individuals (many of whom were experienced, rank-and-file engineers), moved them to a dedicated, single-use location, and spent six months re-architecting our process set with a major focus on the newer CMM Integration (CMMI®) model’s practices. These practices combined Systems Engineering, Software Engineering, Project Management, Organizational Process Improvement, and many support functions. As it turned out, this was money well spent. Sponsorship from upper management never wavered. Lack of sponsorship is, all-too-often, the reason why process improvement efforts fail before they really get started!

Always keep in mind that the process improvement landscape is ever-changing; all failures must be viewed as additional opportunities for improvement and additional data for the Lessons Learned knowledge base.

First, there was the issue of defining our Process Improvement department’s organizational structure. Too little organizational definition can breed chaos whereas too much definition can lead to unneeded bureaucracy. Also, a process improvement organization can not stand separate from the other functional organizations like Engineering and Project Management. Instead, it must co-exist and, even more importantly, be tightly coupled with these other organizational departments. In addition, a good process improvement organization must be allowed to change, adapting itself to meet the ever-changing tactical and strategic objectives of the overall organization.

Case in point: When we began our process improvement efforts in the late 1980’s, we utilized CMM terminology to define our Software Engineering Process Group (SEPG). As we transitioned to CMMI, the SEPG was transformed into the Engineering Process Group (EPG) and other flavors of process groups also came into existence like the Project Management Process Group (PMPG). Since the individual process improvement efforts seemed to be implemented in the typical stove-pipe fashion, our Organization Process Group (OPG) was formed to help coordinate and track the various process improvement activities. Next, understanding that process improvement didn’t stop with Engineering and Project Management, we attempted to bring many supporting functions like Business Development, Business Operations, Finance and Human Resources into the fold under the OPG. This was an abysmal failure mostly due to the fact that these supporting functional organizations were operating several maturity levels below Engineering and Project Management. They were confused by our process improvement speak and overwhelmed by our higher maturity process definition/change processes. So, we reorganized and formed a sub-group (or sister-group) of the OPG called the OPG-Expansion (OPG-E) to facilitate the elevation of the supporting functions’ process maturity levels without slowing down the progress being realized by the OPG within Engineering and Project Management.

A key ingredient in our recipe for success was the implementation of a Process Improvement Support Group, including the Metrics Analysis Group (MAG), which was trained in CMM/CMMI concepts, measurement and analysis techniques (including statistical process control) and process change management. In particular, the MAG’s tasks were to support the definition of measurement and analysis models, develop tools (or modify COTS development products) that supported the measurement collection process and perform and/or support the analysis of data at both the project and organizational levels. We implemented many of the techniques described in J. McGarry, D. Card, et al., Practical Software Measurement, Addison Wesley, 2002, and D. Card, Defect Analysis, Advances in Computers, Elsevier, 2005.

Next, our process document set was revamped, breaking down the process documents into bite-sized morsels that a typical employee could easily swallow. While our previous document set was published in hardcopy, this new architecture was web-based. Also, our organization implemented an integrated approach to our processes, avoiding the usual stove-pipe approach of our previous process document set implementation. In addition, the description of each process had an identical look and feel (no matter which functional department it belonged to) and strict size limits were enforced for each type of document. In addition, the MAG holds annual brain-storming meetings with representatives from engineering, program management, and many of the other support functions to determine exactly what measurements should be deemed important enough to collect and/or analyze. At each meeting, many department representatives entered the room declaring that “We need to reduce the amount of data we were collecting!” Following each meeting, we had AS A GROUP defined 25% to 50% MORE “important” measures and continues on page 6
indicators than we had at the start of the meeting! I will admit that we do occasionally “retire” a particular derived measure or indicator analysis model usually due to nonuse or misuse. We rarely retire any base measure. More likely, the more successfully implemented models, both the process models under instrumentation and measurement collection/analysis models, are continuously updated or improved driven by results from root cause analysis.

Also, our company hired consultants from the then Software Productivity Consortium (now renamed the Systems and Software Consortium, Inc.) as well as PhDs from academia to help re-develop our process set and re-train our employees. After spending millions on this process improvement effort, we decided not to skimp on the deployment aspect and received full support from our company’s upper management.

By the end of 2001, our new process set was deployed and new skills and cultural awareness training sessions were underway in full force. By May, 2002, and after less than one year of “institutionalization; we were assessed at Software CMM Level-4 and at Level-5 by the end of that year. (Note so institutionalization was quite deep in some areas.) The very next year, since we had taken a CMMI rather than CMM based approach, we were able to hold a CMMI SCAMPI Class A appraisal and achieved Level-5 in 2003!

Measurements – Cornerstones for High Maturity Practices

This section describes the high maturity process deployed at BAE, specifically focusing on measurement practices. The staged representation of the CCM associate just four process areas (PAs) with Maturity Levels 4 and 5. These are as follows:

- Quantitative Project Management (QPM)
- Organizational Process Performance (OPP)
- Organizational Innovation & Deployment (OID)
- Causal Analysis & Resolution (CAR)

Since only these four Process Areas (PAs) are required to achieve the Silver & Gold Medals of Process Improvement, CMMI Levels 4 and 5, why does it sometimes seem so difficult and take so long to get there?

![Figure 2. Quantitative Management Process Roadmap](image-url)

that many of the practices we had put in place earlier, but that were not judged sufficient in 2000, continued to be performed

All of these process areas depend on Measurement and Analysis (M&A), a Level 2 process area. Failure to establish the
Measurement Challenge Continued from page 6.

rigor required by M&A at lower levels of maturity makes it difficult to transition to Level 4, which is all about measurement. Bad habits are hard to change.

**Measurement & Analysis**

To start with, you need to have a well-defined and well-implemented measurement collection and analysis process. Figure 2 depicts our current Quantitative Management (QM) Process Roadmap. This simple, 7-step process has realized fantastic results. At CMMI Levels 2 and 3, merely replace the term “QM” in step 1 with the term “Project Management Measurement” and it’s good to go.

Step 1 – First and foremost, you will need to have an understanding of where you currently are, where you think you’re going, and how you expect to get there. In other words, you need to have a plan; or, more likely, many, many plans. Strategic Process Improvement Plans, Organizational/Project Measurement or Quantitative Management Plans, etc. You don’t want to get to the end of the process and be able to tout that you had everything that you needed to accomplish the task at hand except a valid, documented, and agreed-upon plan of action. Process Action Teams (PATs) are utilized to help define, document, instrument and implement the initial development and/or measurement processes.

Step 2 – Now you’re ready to collect measurement data. But beware; there are many hidden perils and pitfalls when it comes to data collection. First, all data has a definite life span and some measurements spoil quicker than others. Measurement may directly influence behavior (through bias and avoidance), especially if the subjects are made painfully aware of the measurement process. Data collection practices should be as transparent to those under instrumentation and as automated as technologically possible. We spent several years designing, building, re-vamping and maintaining our suite of measurement collection tools. In some cases these data collection tools were home-grown, as with our Line-of-Code-Counter Measurement Tool. In other cases, we merely augmented COTS tools, like DOORS or Clear Quest, with additional internal attributes and data collection macros. Either way, much effort was expended during development at deployment of these tools.

Step 3 – Next, you’re ready to store the base measures and create the derived measurements and indicators according to your defined measurement and analysis models. These should have been previously identified and documented in your Measurement or Quantitative Management Plans.

Step 4 – The organization collects the raw data (base and derived measurements) on a regular basis for historic archiving and additional organizational level trend analysis. This data is also the source of the organizational process capability baselines that are necessary to differentiate a “good” measurement from a “bad” one. At lower maturity levels, this data will be used to determine threshold values and at higher maturity levels, this same data are used to define both control and specification limits for process and/or product indicators under more rigorous statistical process control.

Step 5 – Measurement analysis is performed at both the organizational and project levels; the level of rigor and depth of this analysis will vary according to the capability level of each process under review as well as the company’s overall level of process maturity. In Level-5 companies, dedicated Causal Analysis Teams (CATs) are proactive in determining the root causes of process-related issues at both the organizational and individual project levels.

Step 6 – The information and understanding gained from the analysis of measurements is used to drive the decision-making processes.

Step 7 – This is the Process Optimization step, a sort of feed-back loop in terms of process change management. Appropriate corrective actions are undertaken by PATs concerning process definition, maintenance, implementation, instrumentation and optimization for all involved processes, including the development, quality review and data measurement processes.

**Collecting Improvement Information**

Another important consideration for high maturity is the CMMI Generic Practice (GP) 3.2 - Collect Improvement Information. While this is a Level 3 generic practice, the improvement information collected here provides the fuel for CAR and OID. An organization that hasn’t done a good job of improving information collected here provides the fuel for CAR and OID. An organization that hasn’t done a good job of establishing mechanisms for collecting improvement information will be delayed on the road to Level 5.

CMMI General Practice 3.2 (a requirement for a Level-3 organization) involves the collection of historical information that might be useful in future Level-4 and Level-5 activities. This information can include just about anything, and therein lies the problem. Just what are you going to collect?

One commonly-employed method is to dust off the crystal ball, peer deeply into it and see if you can steer it 2 or 3 years into the future to determine what information might be important to you at that time. This method is probably employed as often as is a dartboard when producing software size/effort estimates; and with equally-reliable results.

A slightly more mature method to employ would be to define a living matrix that maps your planned information needs against your business objectives at each CMMI Process Area’s capability level. Table 1 depicts a small portion of our company’s actual information needs which was documented in
an information planning workbook that mapped our information collection strategy against our company's strategic business objectives and standard business processes, as well as at which CMMI Process Area Maturity Level the information might become useful.

Satisfying this particular General Practice can require a bit of an investment. Information collection and maintenance costs time and effort. Most companies operating at lower levels of process maturity are too busy keeping indirect costs down to take the “long-term view.” Besides, even if you had piles of money to throw around, you usually don’t want to spend it collecting information that no one is currently planning to use or even look at. This mapping can help explain the cost and future benefits.

### Typical Measures in an Engineering Environment

When most engineering measurement activities are initiated, one measure that offers many instances for measurement is the Product Quality Defect (i.e., a defect is a part of a work product that does not meet customer requirements). Since the typical Level-2 company has a relatively immature development process set, product defects are quite plentiful and as easy to catch as fish in a barrel.

The peril with focusing too heavily on this particular measure, to the exclusion of other possible quality measures, is that as time passes and you remove more and more fish from the barrel, it becomes increasingly more difficult to find and remove additional fish. Also, as defect prevention activities kick continues on page 9
in, less and less fish will tend to be added to the barrel. Thus, this initially plentiful source of data will sooner or later be “fished out” by design.

Initial Level 2 and 3 quality measure models usually include simple progress tracks of defects discovered, defects resolved, average effort-to-resolve charts, and the like. Figure 3 depicts a typical defect tracking chart.

Additionally, defect categorization using Pareto charts can be very helpful in focusing limited budget at the more prominent issues at hand. Figure 4 depicts a typical project defect Pareto chart.

In addition to defects, other typical (and useful) engineering measures include:
- Requirements Volatility - % of Requirements Added, Modified, Deleted
- Design TBR/TBD Items Burn-Off Rates - TBDs Per Week
- Various Inspection Process & Product Indicators
  - Product Size - Count of Lines Inspected
  - Preparation Rate - Lines Per Preparation Hour
  - Inspection Rate - Lines Per Inspection Hour
  - Inspection Coverage - % of New/Modified Lines Inspected
  - Defect Detection Density - Defects Per 1,000 Lines Inspected
- Software Unit Cost - Average Hours Expended Per Line of Code (Inverse of productivity)
- Testing Failure Intensity - Defects Detected Per X Hours of Testing

Each of the measures listed above, though, has its own special, hidden issues.

For instance, as soon as we started asking projects to track Requirements Volatility, the question arose, “How much is too much?” And I’m not sure that anyone truly knows the answer to that question, or ever will. Certainly, less than 100% volatility is preferable and, realistically, 0% is unattainable (and possibly even counterproductive when attempting to “firm up” the requirements) but who can say what a “good value” range might be. Also, soon after releasing our initial Requirements Stability measurement model, it was pointed out that while “unfunded volatility” was usually “bad,” “funded volatility,” no matter who pays for it, is usually a bonus from a project’s follow-on contract point of view. We were forced to re-think our measurement model, adding an additional measure, namely Funded Requirement Additions, to our Requirements Stability modeling.

In another example, the Software Productivity calculations like the Unit Cost and Productivity Index are both derived measures which are mathematical combinations of two base measures, namely Software Engineering Effort and Logical Lines of Code. However, these base measures can be difficult to accurately instrument, thus increasing the amount of “background noise” in the data. For instance, at our company, all effort data are electronically submitted and employee time charges should total at least 8 hours per day. When calculating something like software productivity, only the hours that are directly applicable to software engineering tasks should be used. However, software engineers are rarely supplied with charge numbers to cover many of the more mundane tasks like answering the continues on page 10
Some measurement models are more sensitive to data noise than others. This is especially true of models built on non-linear relationships like the Testing Failure Intensity curve which exhibits exponential time decay. This model also makes many assumptions, including a stable code set with a fixed number of problems, a fixed starting time and level for the curve, and an accurately recorded amount of testing effort expended. Many of these assumptions are difficult to satisfy. Code cutoff rarely means code cutoff and late code added to the testing baseline after code cutoff is just as likely, if not more likely, to contain defects. Also, testers usually record 8 hours per day against testing regardless of how much time was actually spent testing verses time spent visiting the bathroom or water cooler, writing up problem reports, or talking about last night’s ballgame.

**Predictive Quality Modeling in an Engineering Environment**

In companies with higher maturity measurement and analysis practices, including Quantitative Management (QM), most projects can benefit greatly from Predictive Quality Modeling. This concept truly closes the loop in the QM process by using previously collected and controlled QM-parametric data to calibrate the next run of a model. Our newly-implemented Inspection Planning Quality model uses over 5 years (more than 6,000 Inspections) of historic company Inspection data to help calibrate our previously-implemented Defect Profile Planning model. This new model produces predictions of defect injection and detection profiles as well as estimating the amount of cost avoidance in testing that can be expected based on historic performance, product size, and planned inspection coverages for the various development stages and work products. This new model also allows the quality planner to modify over 50 parameters to play the “What-If” game quickly and easily to determine how best to spend available Inspection budget to maximize inspection defect yield rates.

However, this type of analysis should not be attempted by the faint of heart, nor without the aide of a trained and experienced professional. A serious understanding of the project’s historical performance and the company’s current process performance capabilities are required. Even then, your initial attempts may produce unreliable predictions.

The basic concept is rather simple:

1) Utilize historic performance data to develop predictions of the defect injection/detection rates during the different development and testing phases of project
2) Use that information to predict (and possibly

---

**Figure 4. Project Defect Categorization Pareto Chart**
even affect) the final quality of the product under development.

Our company developed an Inspection Planning Template workbook using MS Excel that guides the project lead through our Predictive Quality Modeling process. At a minimum, the inspection planner needs to enter fourteen critical planning parameters, including the estimated sizes of the product under development and the planned Inspection/Testing Coverage parameters to get a rough prediction of the expected defect injection/detection profiles.

Forty of the 50 adjustment parameters are pre-loaded with company-averaged values. These values can be adjusted by the project to more closely represent their own process capability. The planner can choose to have the workbook predict the required Inspection resources according to the current parameter settings.

After all the parameters have been entered on the Inspection Planning tab, the resulting quality analysis can be reviewed on the Quality Analysis tab. This analysis includes not only the predicted defect injection/detection profiles (see Figure 5), but also the predicted cost savings from the current planned Inspection parameters values as compared to running no product Inspections during development and, instead, planning to capture all defects during the Integration & Testing phases of the project.

Now that a predicted Defect Detection Profile is available, a project can use this data to run in-phase checks on the predicted versus actual defect detection rates. Figures 6 depicts our standard Defect Profiling Plan chart which allows the project to take interim, in-phase Actual-to-Date readings of defect counts and compare them against the Expected-to-Date counts. Analysis of these differences could be useful in determining what may need to be adjusted in either the current development phase or downstream to limit impacts to the final product quality.

In this example, roughly 61% of the code Inspections have been performed but yielded only 45% of the expected defect detections. What might have caused this 16% difference and how shall we adjust our future activities on this product development effort to compensate?

Analysis of the discrepancy between the predicted and actual counts of defects detected in the various development phases allows a project to identify possible quality issues early enough to affect the final outcome.

Should we plan to inspect more of the code product? Should we beef up our Integration Testing team and plan to catch the “missing” defects there? Or maybe the easier code was finished first and inspected early-on with the more complex code (which may be more likely to contain the “missing” defects) inspections yet to be held, so that this anomaly makes perfect sense and nothing needs to be done except attach a call-out bubble explaining the situation.

continues on page 12
Due to the predictive nature of these Product Quality Defect models, the Inspection Plan and the Defect Profiling Plan allow the project lead additional planning of, control over, and insight into the product quality being built into the work product during even the earliest product development phases. Customers on some of our largest contracts (many of them former metrics skeptics, I might add) have found this type of measurement and prediction very interesting. True enough, you may not be very close on your first or even your second attempt. You will, however, learn more about the inner workings of your engineering development cycle, and then after recalibrating the 50 plus parametric values in the model once or twice or thrice, ultimately be able to predict fairly accurately your final product quality as early as the project proposal stage.

One interesting tie among the indicator charts seen in Figures 2 though 5 is that all these widely different views of the Quality Defect measure are created using the same base measurements (i.e., defect counts and associated categorizations). What’s different is the way the measures are organized, filtered, combined and displayed as well as the level of rigor of the associated measurement analysis model employed for each indicator.

Lower maturity measurement and analysis programs often collect too much data and use too little of it. Indicator charts are simple tracks against time with a threshold boundary or moving average thrown in for show. Many, if not most, indicators are post mortem in nature. They are useful as a tool to understand what went wrong after the fact, but not very useful to predict when something might go wrong.

Higher maturity measurement and analysis programs are more predictive in nature. They ask - what can we do to predict when a situation may go awry and do something now to change the outcome for the better? Measurements from one or more defined measures are collected, validated, analyzed for stability/controllability, and data distribution patterns to determine if a predictive behavioral model can be formed for the measurement(s). Control Charts, Histograms, Regression Fits and other statistical methods are employed to gain a deeper understanding of the inner workings of the process(es) under study. Much of this advanced process improvement activity is based on historical measurements that have been previously recorded for “possible future use.”

Measurements & Quantitative Analyses are truly cornerstones in the foundations of High Maturity Practices!

But does this mean that, when your organization is appraised at CMMI Level-5, you’ve reached the end of your process improvement journey?

The short answer is an emphatic “Hopefully not!” continues on page 13
Attaining CMMI Level-5 doesn't guarantee that all process performance issues have been addressed. On the contrary, you probably have a better understanding than ever before of how much room you have in your organization for improvement. Attaining Level-5 means that it has been officially recognized that you have the processes, tools, skills, and other resources and infrastructure in place that are necessary to properly collect, analyze and address these opportunities for improvement.

Level-5 is not the end of the journey, but rather the end of the first leg and the beginning of the rest of the expedition.

Remember, your organization has probably invested millions or even tens of millions of dollars learning how to learn, change and improve. It has also hopefully built up a stable and proven process improvement infrastructure filled with process documentation, process group activities and process training materials. Now that you've got the folks in Engineering, Project Management and maybe even Configuration Management and Quality Assurance under process control, you can turn your process improvement catapults towards remaining organizational bastions of less-than-ideal processes like Business Development, Business Operations and (dare I say it) Travel and Labor Accounting.

About the Authors

Kevin Domzalski is a seasoned member of the process improvement group at BAE Systems National Security Solutions headquartered in San Diego, California, where he currently fulfills the role of organizational process optimization lead, which oversees CMMI Level 5 practices. He joined BAE Systems in 1983 and has served in several capacities in software and systems engineering. He also worked as an automotive industry consultant during a 5-year hiatus from BAE Systems. Domzalski also supports the metrics analysis group (MAG) activities part-time, performing metrics analyses on project and organizational measurements and metrics indicators. He led the MAG activities from 2002 through 2004.

David N. Card is a fellow of Q-Labs. Previous employers include the Software Productivity Consortium, Computer Sciences Corporation, Lockheed Martin, and Litton Bionetics. He spent one year as a Resident Affiliate at the Software Engineering Institute and seven years as a member of the NASA Software Engineering Laboratory research team. Mr. Card received the BS degree in Interdisciplinary Science from American University and performed two years of graduate study in Applied Statistics. Mr. Card is the author of Measuring Software Design Quality (Prentice Hall, 1990), co-author of Practical Software Measurement (Addison Wesley, 2002), and co-editor ISO/IEC Standard 15939: Software Measurement Process (International Organization for Standardization, 2002). Mr. Card also serves as Editor-in-Chief of the Journal of Systems and Software. He is a Senior Member of the American Society for Quality.

Author Contact Information

Kevin Donzalski
BAE Systems
10920 Technology Place
MZ 606-PI
San Diego, CA 92127-1874
Phone: 858-592-5294
Fax: 858-592-5260
kevin.domzalski@baesystems.com

David N. Card
Q-Labs
115 Windward Way
Indian Harbour Beach, FL 32937
Phone: 321-501-6791
Fax: dca@q-labs.com
Estimating System-of-Systems Development Effort

By Jo Ann Lane, University of Southern California

Today’s need for more complex, more capable systems in a short timeframe is leading more organizations towards the integration of existing systems into network-centric, knowledge-based system-of-systems (SoS). Software and system cost model tools to-date have focused on the software and system development activities of a single software system. As we view the new SoS architectures, we find that the effort associated with the integration of these SoSs is not handled well, if at all, in current cost models. USC’s Center for Software Engineering (CSE) began work on an SoS cost model, the Constructive SoS Integration Model (COSOSIMO), in late 2003. This model has evolved using feedback obtained from USC CSE affiliates and other experts in industry and academia.

This paper presents an overview of the COSOSIMO cost model, descriptions of the size drivers and cost factors currently in the model, a summary of recent survey feedback received from USC CSE affiliates and other interested experts from industry, and the impact of survey findings on the current COSOSIMO cost model. It concludes with future plans for the COSOSIMO model.

History of COSOSIMO

Why a COSOSIMO? We are seeing a growing trend in industry and DoD to “quickly” incorporate new technologies and expand the capabilities of legacy systems by integrating them with other legacy systems, Commercial-Off-the-Shelf (COTS) products, and new systems. With this development approach, we see new activities being performed to define the new architecture, identify sources to either supply or develop the required components, and then to integrate and test these high level components. Along with this “system-of-systems” (SoS) development approach, we have seen a new role in the development process evolve to perform these activities: that of the Lead System Integrator (LSI).

Today, there are fairly mature tools to support the estimation of the effort and schedule associated with the lower-level SoS component systems. For software development activities, there are the COCOMO II, Cost Xpert, Costar, PRICE S, and SEER-SEM cost models. At the single system level, there is COSYSMO to estimate the system engineering effort and PRICE H and SEER-H to estimate hardware development costs. For COTS implementation and integration, there is CO-COTS to estimate the effort associated with the assessment, tailoring, and glue-code implementation of COTS software products. [1] However, none of these models includes LSI activities such as the definition of the SoS architecture, the solicitation and procurement process for the SoS components, and the integration of the SoS components into the SoS framework. Many LSI organizations often estimate these costs using a percentage of the lower-level system component development costs. As we see more and more of this type of development, it is important to get a handle on such questions as:

- How much should an organization budget for SoS integration activities?
- Is an extra 10%, 20%, or 50% sufficient? Too much?
- What factors or characteristics make actual effort higher or lower?

COSOSIMO is a parametric model to compute just this effort. It is designed to estimate the up-front LSI effort associated with SoS abstraction, architecting, source selection, and systems acquisition, as well as the effort associated with the later activities of integration, test, and change management. The goal is to support estimation activities for estimating the LSI effort in a way that allows users to develop initial estimates and then conduct tradeoffs based on alternatives. With the addition of COSOSIMO to the COCOMO suite of estimation tools, there will be more complete coverage of the development activities associated with implementing a system-of-systems.

What is an SoS? Key to developing a cost model such as COSOSIMO is understanding what a “system-of-systems” is. Early literature research [3] and workshops with industry affiliates [4] shows that the term “system-of-systems” means many things to many different people and organizations. In the business domain, it is the enterprise-wide and cross-enterprise integration and sharing of core business information across

Figure 1. Sample SOS continues on page 15
Estimating System-of-Systems

Continued from page 14.

In the military domain, it is a dynamic communications infrastructure to support operations in a constantly changing, sometimes adversarial, environment. For some, an SoS may be a multi-system architecture that is planned up-front by an LSI. For others, a SoS is an architecture that evolves over time, often driven by organizational needs, new technologies appearing on the horizon, and available budget and schedule. The evolutionary SoS architecture is more of a network architecture that grows with needs and available resources.

In any of these cases, users and nodes in the SoS network may be either fixed or mobile. Communications may be either point-to-point or broadcast. Networks may tie together other networks as well as nodes and users. SoS component systems typically come and go over time. These component systems can operate both within the SoS framework and independent of the SoS framework.

Andrew Sage and Christopher Cuppan best summed it up when they state that SoS' typically contain a majority of the following characteristics: operational independence of individual systems, managerial independence of individual systems, geographic distribution of SoS component systems, emergent SoS behaviour not contained in any one component system, and evolutionary development over time, with functionality continually being added, deleted, and modified. [9]

What is an LSI? After defining the set of SoSs to be supported by COSOSIMO, the focus turned to defining an LSI in the SoS environment. In order to answer questions such as “how much effort” and “how long”, it is important to first understand the types of activities performed in SoS architecture development and integration and how these vary across different SoS implementations. Sources reviewed included LSI statements of work (or summaries of statements of work) that could be accessed on the web for new or currently on-going LSI efforts. As a result the COSOSIMO team came up with the following description [7]:

- An LSI is an organization (or set of organizations) selected to oversee the definition, development and integration of an SoS.
- Typical LSI activities include:
  - Concurrent engineering of requirements, architecture, and plans
  - Identification and evaluation of technologies to be integrated
  - Source selection of vendors and suppliers
  - Coordination of supplier activities
  - Validation and feasibility of SoS architecture
  - Continual integration and test SoS-level capabilities
  - On-going change management at the SoS level and across the SoS-related Integrated Product Teams (IPTs) to support the evolution of requirements, interfaces and technology.
- Typically LSIs do not develop system components to be integrated. The one possible exception is the development of the SoS infrastructure into which the system components will be integrated.

COSOSIMO Structure

For the purposes of cost modelling, it became clear early on that a single cost model could not support the variety of SoSs. As a result, efforts were initiated to better define the set of SoSs that would be supported by COSOSIMO [8]. The key characteristics that identified the set of SoSs well-suited for cost modelling were:

- Strategically-oriented stakeholders interested in tradeoffs and costs
- Long-range architectural vision for SoS
- Developed and integrated by an LSI
- System component independence.

COSOSIMO Parameters. Once the key discriminators or characteristics of the SoS and LSI were determined, efforts could focus on defining the cost model size drivers and scale factors, illustrated in Figure 2. The cost and size drivers are based on the SoS architecture characteristics (product), processes used for design and integration/test (process), and LSI experience and capabilities (personnel).

continues on page 16

The selected size drivers focus on quantifiable sizes associated with the SoS interface protocols and the political complexities of negotiating agreements with other organizations to support the required system component changes needed to enable the SoS protocols [6]. They are:

- Number of SoS Interface Protocols: The number of distinct interface protocols to be provided by the SoS framework.
- Number of Independent System Component Organizations: The number of organizations that are providing system components that will operate within the SoS framework.

Each of these size inputs is weighted with respect to expected complexity. The interface protocol complexity is determined by factors such as number of protocol layers, desired security, and whether this is a new protocol being implemented for the first time or it is an existing protocol. The independent system component organization complexity is determined by the expected level of cooperation between organizations and the competing needs of the SoS versus the needs and schedules of the independent component system provider or sponsor.

The early-design scale factors have been selected to capture the affects of key development processes and business/political decisions [6]. They are:

- SoS Architecture Maturity: A parameter that represents the level of maturity of the SoS architecture. It includes the level of detail of the interface protocols and the level of understanding of the performance of the protocols in the SoS framework.
- Cost/Schedule Compression: The extent of business or political pressures to reduce cost and schedule.
- Integration Risk Resolution: A multi-attribute parameter that represents the number of major integration risk items, the maturity of risk management and mitigation plan, compatibility of schedules and budgets, expert availability, tool support, and level of uncertainty in integration risk areas.
- Component System Maturity and Stability: A multi-attribute parameter that indicates the maturity level of the system components (number of new component systems versus number of component systems currently operational in other environments), overall compatibility of the system components with each other and the SoS interface protocols, the number of major component system changes being implemented in parallel with the SoS framework changes, and the anticipated change in the component systems during SoS integration activities.
- Component System Readiness: Indicates the readiness of component systems for integration. The user evaluates level of verification and validation that has/ will be performed prior to integration and the level of subsystem integration activities that will be performed prior to integration into the SoS integration lab.
- Integration Team Capability: Represents the anticipated level of integration team cooperation and cohesion, integration personnel capability and continuity, and integration personnel experience with the application, language, integration tools, and integration platform(s).
- Maturity of the Integration Processes: A parameter that rates the maturity level and completeness of an integration team’s integration processes, plans, and the SoS integration lab.

How Does COSOSIMO Compare to the CSE System Engineering Cost Model? During the development of COSOSIMO, CSE industry affiliates have questioned whether COSOSIMO is really all that different than the system engineering cost model, COSYSMO. Many thought that the LSI effort associated with a SoS development was just a different “system of interest” as defined in [2]. So, in July 2005, several CSE industry affiliates participated in a survey to further investigate this view and to better understand the key activities performed on SoS LSI programs and the size drivers and scale factors related to both traditional system engineering and SoS LSI programs [8]. The analysis of the survey responses strongly suggests that:

a. The COSYSMO size drivers and scale factors are generally thought to be applicable to the effort associated with LSI technical activities. However, some additional scale factors and an SoS-unique calibration may be required to capture the effects of program aspects more unique to the very large LSI efforts.
b. The COSYSMO model size drivers and scale factors are not sufficient to estimate the management effort associated with LSI programs. Therefore, if COSYSMO is used to estimate the LSI technical effort, an additional program management cost model may be required to estimate the corresponding LSI management effort.

Future Work. COSOSIMO efforts continue on several fronts:

- Collaboration with key personnel on LSI projects to better understand the key drivers and influences of various project characteristics and dynamics
- Collection of additional survey responses to better understand LSI activities and associated size drivers and scale factors
- Collection of actual data from SoS development increments or iterations to start the calibration of the cost model scale factors

continues on page 17
Estimating System-of-Systems

Continued from page 16.

References


About the Author

Ms. Jo Ann Lane is currently a PhD student at USC working in the area of System Architecting and Engineering and teaches software engineering at San Diego State University. Prior to joining the USC PhD program, Ms. Lane was a key technical member of Science Applications International Corporation’s (SAIC) Software and Systems Integration Group. She has over 28 years of software engineering and development experience in software development, software project management, software process definition and implementation, and metrics collection and analysis. Ms. Lane earned her BA degree in Mathematics and her MS degree in Computer Science from San Diego State University.

Author Contact Information

Jo Ann Lane
University of Southern California
Center for Software Engineering
Los Angeles, CA 90089
Phone: (858) 945-0099
jolane@usc.edu
Measurements: Managing a Large Complex Modernization Effort—While Protecting Your Program from the “Katrina Factor”

By Stephen C. Hawald, Robbins Gioia

In any software development project, it is absolutely critical to measure the program and projects/releases for cost, schedule and scope. There is nothing new here! However, most fledgling software development program managers or program managers miss the planning step for a strong framework of “levees” to help manage the project from all angles. Reviews and analysis, ideally performed weekly, can pinpoint breaches in the “levees” early on.

The other areas of the measurement framework often missed or poorly reported are desired business outcomes, customer satisfaction levels, return on investment objectives, process measures that include quality assurance reviews and outcomes, corrective actions reports, cost of rework, deliverable and work product rejection rate.

How do we develop a measurement framework? It is more like adopting a proven methodology such as the Practical Software and Systems Measurement (PSM) approach. Select the version that communicates easily with your organization and tailor it for your needs—you do not need to reinvent the wheel to “mature” your organization. Before a PMO gets started on its measurement end-state, the measurements the PMO staff gets are not the ones they actually need. Apart from inadequate measures, most PMOs have too much of the wrong data, problems with the data collection system, duplicate data, dirty data, and an expensive measurement repository. Unfortunately, until fairly recently a performance measurement plan was not one of the squeaky wheels, so it tended to go unnoticed until it was too late. With your schedule off by more than 15 percent, you will almost never be able to recover in a timely manner when the measurement city fills up to capacity and the program is overflowing the data measurements levees. You now have everything that has been developed over the last five years of your software program floating by from the “data measurement lake” and start to wonder how you’re going to clean up this mess to return to meaningful data on your measurement program.

Planning is the most critical step and it almost always takes longer to complete than anticipated—but planning ahead at the beginning for the end-state will save you potentially hundreds of hours of rework, particularly if you are on the road to “mature” your organization. Before a PMO gets started on developing key performance measures, it needs to determine and analyze its information needs by precisely defining what is to be measured and then how these measures should be analyzed. The what can be discerned by a thorough analysis of the formalized requirements of the project—which, if the project is well chosen, will align with the centralized objectives of the organization. As always, strong executive leadership and sponsorship roles are the keys to a well-written, holistic successful measurement plan.

My recommendation would be to develop a data measurement plan. The Katrina Factor

If there is one thing almost all projects have in common, it is inadequate planning. This shortfall is particularly apparent in all areas of planning program and project measures. They are frequently collected by project management office (PMO) staff as a harried afterthought in response to customer or government requirements by way of contract statement of work as well as government regulations. If you are awarded a federal government contract then you need compliance in areas such as the Government Performance Results Act (GPRA), CFO Act, Disability Act; AKA-Section 508, President’s Management Agenda, Financial Management Integrity Act, OMB 300, and Sarbanes-Oxley Act. It is an understatement to say that these requirements are not ones to be taken lightly. Typically, the continuation of the project depends on meeting them successfully.

Few PMO organizations invest enough resources, time, and money into developing meaningful measurements. As a result, their measurements are useless most of the time. You may ask yourself, if I am working in a PMO with my Project Management Body of Knowledge (PMBOK®) guide firmly in hand, what could possibly go wrong? The answer is that project and program managers are more concerned with putting out today’s fires than some theoretical end state. Simply put, we need to get through today with our customer, product manager, and PM director and stay alive for tomorrow’s next surprise package on the project. Yes, we see the “data storm” on our radar, but it does not seem as pressing as all the top 10 projects on our to-do list.

To prevent the Katrina Factor and ensure forward-looking, intelligent, responsive, and effective means of collecting and measuring data, consider these three key points in reinforcing your data/measurement levees to avoid a disaster in your program measurement city.

1. Plan for the Storm—Find and Align Your Measurement End-State

A project’s life requires countless important activities demanding immediate attention, especially when new. Most measurements are done as the organization needs them. The more measures, the merrier the PMs are. But ultimately their merriment turns to gloom. Because there is no plan for or strategic view of how measures will be needed in the project’s end state, the measurements the PMO staff gets are not the ones they actually need. Apart from inadequate measures, most PMOs have too much of the wrong data, problems with the data collection system, duplicate data, dirty data, and an expensive measurement repository. Unfortunately, until fairly recently a performance measurement plan was not one of the squeaky wheels, so it tended to go unnoticed until it was too late. With your schedule off by more than 15 percent, you will almost never be able to recover in a timely manner when the measurement city fills up to capacity and the program is overflowing the data measurements levees. You now have everything that has been developed over the last five years of your software program floating by from the “data measurement lake” and start to wonder how you’re going to clean up this mess to return to meaningful data on your measurement program.

Planning is the most critical step and it almost always takes longer to complete than anticipated—but planning ahead at the beginning for the end-state will save you potentially hundreds of hours of rework, particularly if you are on the road to “mature” your organization. Before a PMO gets started on developing key performance measures, it needs to determine and analyze its information needs by precisely defining what is to be measured and then how these measures should be analyzed. The what can be discerned by a thorough analysis of the formalized requirements of the project—which, if the project is well chosen, will align with the centralized objectives of the organization. As always, strong executive leadership and sponsorship roles are the keys to a well-written, holistic successful measurement plan.

My recommendation would be to develop a data measurement plan.

continues on page 19
Measurements: Continued from page 18.

Example of a Program Measurement Integrated View

![Image of a Program Measurement Integrated View]

The measurement plan and vision (DMPV) to align with your organization's strategic plan and vision. From the DMPV, the data/measurement framework (DMF) can be developed for all the components for your program and software releases. The program measurement integrated view can display the view as shown in Figure 1.

The measurement views that are tailored to fit your program needs should include a measurement e-dashboard to display key functional areas, such as performance measures, business outcomes/customer satisfaction measures, and process measures, including quality assurance and corrective actions and defects. (See Figure 2 for an example of a measurement e-dashboard.) These are the areas you need to target to acquire quality data and the right kind of reporting.

- Performance measures. These measures tell you how efficient you are in getting what you expect from the project. One very useful performance measure is earned value management (EVM), which integrates project cost, schedule, and performance. Simply put, EVM determines if you are getting what you pay for as far as project completion compared to project payments. EVM techniques are critical for all cost plus (CP) contracts and task orders.
- The measures you select are ultimately more important than the tool. Use caution. There are a number of new measurement tools available that are both inexpensive and expensive to use on large projects. High-end tools used on small projects and programs can be overkill and complicated to use. Most PMOs have problems collecting event-level data—but it is possible—and certainly more efficient—to collect measures “painlessly” as a side effect of the work. There are many management commercial-off-the-shelf (COTS) and proprietary management tools for programs and projects that can assist with this capture by keeping track of deliverables, work products, releases, artifacts, code, and acceptance dates.
- Business outcomes/customer satisfaction measures. These measures tell you how effective the program is...
Measurements: Continued from page 19.

in regard to the end product and the internal/external customer. Measurements can be obtained from customer satisfaction surveys and before-and-after data. An example might be measuring how quickly and effectively a new software release, drop, or package works for the end user or customer.

- Process measures. Examples of these measures include the acceptance rate of work products or deliverables—and conversely, the number of rejections and packages for rework. This effort can be accomplished by applying the use of a modified PSM workshop tailored for the PMO process assets (processes on how work is done) to the needs of the program with decision makers and PMO business owners.

These three areas are necessary to the development and implementation of consistent, comprehensive measures of customer satisfaction, financial quality and financial performance, as well as software builds for compliance and number of reworked packages.

2. Strengthen the Levees—Select and Use Guiding Key Principles.

Adopt relevant guiding principles, such as a model from the Software Engineering Institute's Capability Maturity Model® Integration (CMMI) or the International Organization for Standardization (ISO) model. A measurement methodology should be one of these guiding principles, either in place when the project is begun or soon adopted with stakeholder buy-in. PSM is a methodology commonly used in measurement programs and ties into the CMMI version 1.1 and the upcoming version 1.2.

Apply Six Sigma techniques in high-maturity PMO organizations to analyze data for errors and deliverable rejections. Data must be synchronized to ensure its integrity.

Figure 2. Sample Dashboard

Keep the dashboard top web page clean and easy to read with fewer than 15 measurements. Executives can drill down for lower-level details. Plan for measurement reporting and push/pull data.
This effort will be required for accomplishing CMMI Maturity Levels 2, 3, 4 and especially Level 5, if this level can be justified for your line of business and products, such as pharmaceuticals.

Create and use common standards for data measurement, collection, analysis, and reporting. Keep it high-level for executives (for example, use a green/yellow/red stoplight chart), but enable a drill-down capability to the detail data as well as analysis results.

Document and establish a target for each measure. It is very difficult to analyze a measure if there is no target or baseline to measure against. Targets can be developed from industry standards such as EVM; models such as COCOMO (a software cost model); company goals such as market share percentage; schedule and milestone dates; or any other methodology suitable to the specific type of measure. Develop a measurement handbook for users as web pages dynamically linked to the measurement dashboard for ease of maintenance.

Realize that the process of defining and analyzing project measures has to be exhaustively pursued for each project or software release. It is counterproductive to use old measures without conducting an analysis of what the customer needs or neglecting to validate your data by keeping your analysis to yourself.

3. It’s Not Enough to Keep Your Finger in the Dike—Perform Continuous Improvements

The measurement journey is never done. To avoid the risk of your project standing still or even sliding backward, recurring or other trouble areas of management or technical problems need to be addressed by appropriate data. Use all the tools and techniques you can find to correct rejection rates or errors. Each “city” or software program will be unique in its own right, so tailor and adopt best practices for continuous improvements that best fit your PMO—and you will avoid drowning in the measurement data lake!

If all these steps could be summed up into a sentence, it would be to invest quality, people, time, and money into developing meaningful measurement specifications—and PLAN. Then you will have PMO measurement programs that support the strategic business and mission objectives and information needs of your project and your organization.

About the Author

Steve Hawald joined Robbins Gioia in 2003 as an executive consultant with over 20 years of experience in all areas of technology from technical consulting to CIO. He developed and is currently the solution champion for the new Robbins Gioia practice solution area called Process Refinement and Optimization (PRO) for process improvements services supporting SEI CMMI and CMMI-AM, lean and six sigma, balanced scorecard, enterprise risk management, measurements and quality management. He has published several recent articles relative to these areas.

He has a B.A. from the University of Maryland at College Park, Maryland, in Accounting and Engineering, two master degrees from The American University in Technology Management and The University of Baltimore in Accounting, and has completed his Ph.D. course work at George Mason University in Information Technology. Steve recently served on the Post Secondary Education Council's (PESC) XML Forum Steering Committee and past board member on the Capital Area Chapter for the Society of Information Management (SIM). He is a Six Sigma Black Belt, a Certified Information Systems Manager (CISM) and a Certified Project Management Professional (PMP).

e-mail: stephen.hawald@associates.dhs.gov
phone: 703-650-3235
Work address: Robbins-Gioia LLC
1801 Beauregard Street C-3174
Alexandria, VA 22311
Building Systems Using Software Components

By Robin Ying, Stevens Institute of Technology

Introduction

It is human nature that when facing a complex problem, we tend to break it into pieces and solve the smaller problems. This divide-and-conquer strategy works for many different situations, all the way from Napoleon’s military movements to building software systems. When constructing software, the key issue is designing the software architecture so that the entire system is properly decomposed into parts or components. The parts or components can be anything from subroutines or code modules in programs to complete sub-systems that form separate executables.

The famous computer scientist, Edsger W. Dijkstra, once raised an interesting argument: Suppose a program is composed of N parts, and each part has a probability of correctness of \( p_i \). Then as the parts are assembled together, the correctness of the whole system will be

\[
P = \prod_{i=1}^{N} p_i
\]

If \( N \) is large, then unless each \( p_i \) is very close to one, \( P \) will be close to zero. At first glance, it may seem that Dijkstra’s argument is against decomposition, since as \( N \) increases, \( P \) decreases. In practice, as the complexity of a system increases, there is an increasingly smaller chance we can achieve correctness without flaws. However, by breaking the problem into smaller pieces, we allow ourselves to deal with a less complex situation, and thus the chance of achieving overall system correctness increases. Ideally, if each component works flawlessly (i.e. \( p_i = 1 \)), then \( P = 1 \) regardless of the value of \( N \). Wouldn’t this be wonderful? In theory, all we have to do is to break down the program into many tiny parts, each one contains no more than a few lines of code. We would then ensure individual component correctness, and thus overall system correctness.

However, as we know again, this does not work in practice. Even with each individual component having a correctness of 1, when the pieces are assembled together, we introduce integration errors. As the number of components in a system increase, the number of integration errors increase as well.

Thus, the actual process of software system construction is a compromise between these two extremes. We decompose the system into “manageable” components, and try to minimize the integration error as we assemble them together. This requires that the software architecture be carefully designed to achieve this delicate balance.

Modern software architecture practices promote “reuse”: that is, to build reusable components that can be utilized in the project multiple times, or by other systems. This produce-once-use-many-times strategy is a great way to increase efficiency and reduce costs. It also complements the component-based architecture. Very often, the reused components are treated as “building blocks”. The current trend of component-based architecture is to incorporate as many COTS (commercial off-the shelf) components as possible to reduce the development time and costs. However, choosing the proper COTS components and carrying out the integration carefully are challenges in the new dimension.

What’s the Problem?

As previously mentioned, integrating components generates errors. Let’s assume a software system has two components: A and B. Integrating A and B produces the whole system \( S \).

In component A, the units used in computation are English units (inches, miles, ounces, pounds, etc.) and in component B, the units are in metric (centimeter, kilometer, grams, kilograms, etc.). To date, there is no programming language that can automatically distinguish the floating point number “1.25”, a basic data type, to mean 1.25 inches or 1.25 centimeters. A careless integration of A and B can produce devastating errors, illustrated in the NASA Mars Climate Orbiter disaster on September 23, 1999. In the NASA incident, an error in the altitude calculation caused the multi-million dollar satellite to be destroyed by the planet’s atmosphere. The error report revealed that unit mismatch was one of the major causes of the mission’s failure.

Careless reuse can be another source of disaster. The Therac-25 radiation therapy machine accidents are an example of this type. From June 1985 to January 1987, six deadly accidents occurred due to machine malfunction, causing the patients to be exposed to radiation overdoses. In this case, the Therac-25 manufacturer reused the control software written for earlier models, which used electro-mechanical interlocks to prevent radiation overdose. In the Therac-25, the electro-mechanical interlocks were removed and no proper updates on the reused software were made to provide the needed safety check.

In today’s object-oriented software design approach, we initially develop a structure where multiple parts are at a roughly equal level of abstraction. Then we analyze and refine each part continues on page 23
so that they will fit together and achieve the desired features and functions before starting the implementation.

Experience tells that it is often better to organize a system around data than around functions. The key consideration in decomposing a system is how we couple the components. We need to minimize coupling between the components and maximize the cohesion within each component so that each component can be built independently. In addition, this approach also provides the better reusability of a component, since the component may be used for systems completely unrelated to the original.

Decades ago, the Shell scripts (an interpreted programming language in the UNIX system) together with the UNIX commands and environment achieved a set of desired characteristics for component-based software programming. Today, the Java system (programming language and the Java VM) and C#.Net from Microsoft are taking this concept further, by providing more building-block components, stronger type checking, and more built-in safety checks (e.g. array out of bound check). This higher level of abstraction incorporates the component-based design concept into the programming languages. It “componentizes” system-level utilities and enhances their reusability. This not only makes the lives of software developers easier, but also greatly increases the quality of the resulting system.

Components and Trustworthiness

By definition, a software system is said to be trustworthy if the system is safe, reliable and secure. Therefore trustworthiness of a software system includes three quality attributes: reliability, security and safety. These attributes not only provide for system stability, but also for ethical responsibility when dealing with systems that may affect people’s lives. Let’s discuss how the components in a software system affect these quality attributes.

1-Reliability

Reliability is one of the most important non-functional requirements for all systems. The reliability of a software system depends on many factors, such as the computer hardware, the operating system, the physical environment, but most importantly, the software system itself. The reliability of a software system should be judged holistically, i.e. after all, its components are integrated together, not by looking at each component individually.

Let’s look at a simple example. System $S_1$ consists of components A and C, and system $S_2$ consists of components B and C, where component C performs arithmetic calculation on data passed from component A in $S_1$ and B in $S_2$. C does not perform the divide-by-zero check and this fact is not revealed in its specification. Component A performs its own divide-by-zero check before passing data to C, but B does not. Assuming no other complicating factors, then we can say that $S_1$ is reliable and $S_2$ is not. In addition, component C is not reliable since it does not reveal the fact that it does not provide a divide-by-zero check in its specification.

In addition, the Mars Climate Orbiter disaster shows that a single integration error can make the whole system unreliable even if all its components are fully reliable. Nevertheless, we always want to start with a set of reliable and reusable components; as said in a Chinese proverb: a good start is half the success.

The reliability of a software system can be improved by rejuvenation. Software rejuvenation is a preemptive, periodic restart of a running system at a clean internal state to prevent future failures. When a system is composed of multiple components, do we need to rejuvenate all of them? The answer lies in the question, “How close are these components coupled together?” Consider a web-based application that has an application server and a database server. The application server may need the rejuvenation treatment while the database server may not, or both may need rejuvenations, but at different frequencies. This also shows that if we design a system to be composed of multiple loosely coupled components, where each component can be rejuvenated independently, we have a better chance at improving its reliability. However, rejuvenation incurs overhead, such as server downtime or inaccessibility of data. Oftentimes, statistical modeling methods are used to determine the optimal frequency for rejuvenation.

2-Security

Achieving the desired security needs cooperation of the software system, the operating system, and the running environment. In addition, one must consider both software security as well as physical security. Operating systems without kernel memory protection can never support any secure application properly. Top-level security requirements from the government demand physical air gaps separating the computer networks and vault-like rooms to secure computer equipment.

The security of a software system is usually handled by a set of its components. These components, whether they are from COTS or custom developed, handle the access authentication, intruder detection and prevention functions, and other needed features. These components usually have high degree of reusability. Failure at the component level will often compromise the security of the entire system. Thus, the design and construction of these security components have evolved into a specialty. Today, many well-designed COTS components with detailed specifications are available for use. However, proper integration is still the key to ensure that these components are being used in the manner they were designed.

Thus, there is no common rule that relates the overall software system reliability to the reliability of its components.

Building Systems  Continued from page 22.
The safety of a software system includes its interaction with the environment, as well as other affected groups such as people or animals. An unsafe system means that it may cause harm to those involved and raise concerns of varying degrees. Furthermore, if the software system is used to control physical devices, such as machines, the safety of the system is of utmost importance.

Safety requirements are non-functional requirements; however, their importance often supercedes functional requirements. In many cases, functional requirements need to be compromised or re-evaluated in order to bring the overall system within the safety margin.

Although safety related features might be handled by a set of components, similar to reliability and security, the safety of the entire system is an integration issue. This even extends to the operators – those who use the system. If the usage of the system is out of the design intention or limitation, severe hazardous conditions may occur and render the system to be unsafe.

The following table exhibits the software control categories as defined in the Military Standard System Safety Program Requirements, MIL-STD-882C.

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous</td>
<td>Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Inappropriate software action or failure to act can contribute directly to a Top Level Mishap (TLM).</td>
</tr>
<tr>
<td>Semi-Autonomous</td>
<td>Software displays safety-related information or exercises control over potentially hazardous hardware systems, subsystems, or components with the possibility of intervention by independent safety systems to preclude the occurrence of a TLM. However, the possibility of intervention by itself is not considered sufficient to prevent a mishap.</td>
</tr>
<tr>
<td>Semi-Autonomous with Redundant Backup</td>
<td>Software displays safety-related information or exercises control over potentially hazardous hardware systems, subsystems, or components. However, there are two or more independent safety measures within the system, but external to the software item, which mitigates the possibility of leading to a TLM.</td>
</tr>
<tr>
<td>Influential</td>
<td>Software processes safety-related information but does not directly control potentially hazardous hardware systems, subsystems, or components.</td>
</tr>
<tr>
<td>No Safety Involvement</td>
<td>Software does not process safety-related data or exercise control over potentially hazardous hardware systems, subsystems or components. The software cannot contribute to a mishap.</td>
</tr>
</tbody>
</table>

### 3-Safety

The safety of a software system includes its interaction with the environment, as well as other affected groups such as people or animals. An unsafe system means that it may cause harm to those involved and raise concerns of varying degrees. Furthermore, if the software system is used to control physical devices, such as machines, the safety of the system is of utmost importance.

Safety requirements are non-functional requirements; however, their importance often supercedes functional requirements. In many cases, functional requirements need to be compromised or re-evaluated in order to bring the overall system within the safety margin.

Although safety related features might be handled by a set of components, similar to reliability and security, the safety of the entire system is an integration issue. This even extends to the operators – those who use the system. If the usage of the system is out of the design intention or limitation, severe hazardous conditions may occur and render the system to be unsafe.

The following table exhibits the software control categories as defined in the Military Standard System Safety Program Requirements, MIL-STD-882C.

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous</td>
<td>Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Inappropriate software action or failure to act can contribute directly to a Top Level Mishap (TLM).</td>
</tr>
<tr>
<td>Semi-Autonomous</td>
<td>Software displays safety-related information or exercises control over potentially hazardous hardware systems, subsystems, or components with the possibility of intervention by independent safety systems to preclude the occurrence of a TLM. However, the possibility of intervention by itself is not considered sufficient to prevent a mishap.</td>
</tr>
<tr>
<td>Semi-Autonomous with Redundant Backup</td>
<td>Software displays safety-related information or exercises control over potentially hazardous hardware systems, subsystems, or components. However, there are two or more independent safety measures within the system, but external to the software item, which mitigates the possibility of leading to a TLM.</td>
</tr>
<tr>
<td>Influential</td>
<td>Software processes safety-related information but does not directly control potentially hazardous hardware systems, subsystems, or components.</td>
</tr>
<tr>
<td>No Safety Involvement</td>
<td>Software does not process safety-related data or exercise control over potentially hazardous hardware systems, subsystems or components. The software cannot contribute to a mishap.</td>
</tr>
</tbody>
</table>

### References


### About the Author

Dr. Ying began his software work at Bell Laboratories. He was among the first few to start telecommunication business application software development using C language and UNIX system in the early 80s. Since then he has been an active practitioner in all aspects of the software development process. He received his Ph.D in Electrical Engineering and Computer Sciences from University of California at Berkeley. He is currently the Dean of Technology and Learning Services at the Imperial Valley College, and an adjunct faculty member of the WebCampus, Stevens Institute of Technology.

Email at Imperial Valley: robin.ying@imperial.edu

Email at Stevens Institute: rying@stevens.edu

Detailed Bio: http://webcampus.stevens.edu/info/faculty3.html#ying
The first 50 people to send in a completed survey will receive a FREE DoD/IT Acronym CD from the DACS.

This valuable CD-ROM contains over 9,000 Department of Defense and Information Technology acronyms. There are hundreds of acronym lists available but none are as well done as this CD AND specifically targeted towards DoD and Information Technology. This unique-shaped CD-ROM plays in your computer’s regular, hub-mounted, CD drive. You’ll use this great resource over and over again. It’s FREE, just for filling out our brief survey on the next page!

Data & Analysis Center for Software (DACS)

http://iac.dtic.mil/dacs/
# Software Tech News Subscriber Survey

1. Which volume of the Software Tech News did you receive?  

2. When did you receive the newsletter? (month/year)  

3. How satisfied were you with the CONTENT of the newsletter? (Article Quality)  

   - Very Satisfied  
   - Satisfied  
   - Neither Satisfied nor Dissatisfied  
   - Dissatisfied  
   - Very Dissatisfied  

4. How satisfied were you with the APPEARANCE of the newsletter?  

   - Very Satisfied  
   - Satisfied  
   - Neither Satisfied nor Dissatisfied  
   - Dissatisfied  
   - Very Dissatisfied  

5. How satisfied were you with the OVERALL QUALITY of the newsletter?  

   - Very Satisfied  
   - Satisfied  
   - Neither Satisfied nor Dissatisfied  
   - Dissatisfied  
   - Very Dissatisfied  

6. How satisfied were you with the ACCURACY of the address on the newsletter?  

   - Very Satisfied  
   - Satisfied  
   - Neither Satisfied nor Dissatisfied  
   - Dissatisfied  
   - Very Dissatisfied  

7. Approximately how much of the newsletter do you read?  

   - The entire issue  
   - Most of the content  
   - About half the content  
   - Briefly Skimmed  
   - Didn’t Read  

8. Would you read this newsletter in an E-mail newsletter format?  

   - Definitely  
   - Probably  
   - Not Sure  
   - Probably Not  
   - Definitely Not  

9. How did you request the product or service?  

   - Phone Call  
   - E-mail  
   - DACS Website  
   - Subscription Form  
   - Other  

10. Would you recommend the Software Tech News to a colleague?  

    - Definitely  
    - Probably  
    - Not Sure  
    - Probably Not  
    - Definitely Not  

11. What topics would you like to see this newsletter devoted to?  

    **Comments (Optional)**

    **Contact Information (Optional*)**

    **Name:**  
    **Position/Title:**  
    **Organization:**  
    **Office Symbol:**  
    **Address:**  
    **City:**  
    **State:**  
    **Zip Code:**  
    **Country:**  
    **E-mail:**  
    **Telephone:**  
    **Fax:**  
    **Functional Role:**  
    **Organization Type:**  
    - Air Force  
    - Army  
    - Navy  
    - Other DoD  
    - Commercial  
    - Non-Profit  
    - Non-US  
    - US Government  
    - FFR&D  
    - Other  

*Note: You must give us your address to receive the CD.
Great Software Tools from the DACS!

Return On Investment from Software Process Improvement Products CD

New XML Database Included!

ROI/SPI CD:
- Technical Report & Spreadsheet Model
- ROI Database
- Cost Benefit Analysis

SW Reliability Sourcebook CD:
- Covers the Software Life Cycle
- Hundreds of Hotlinks & Embedded Documents

Purchase these tools at the DACS Store
www.dacsstore.com
or contact
800-214-7912  dacs@dtic.mil

http://iac.dtic.mil
Are you getting **All** you can from your software investment?

The DACS ROI Dashboard

**Impact on Quality (% defect reduction)**

- 75th Percentile: 90
- Oklahoma City Air Logistics Center, Test Program Set and Industrial Automation line observed the defect density decrease 91%, from 3.3 to 0.3
- Defects per KSLOC, while achieving CMM Level 4.

**Technologies Covered:**
- SEI CMM/CMMI
- SEI Team Software Process (TSP)
- SEI Personal Software Process (PSP)
- Inspections
- Reuse
- Cleanroom
- And Many More!

**Graphs Showing Impact Of Software Technologies On:**
- ROI
- Productivity
- Quality

**Summarizes Facts From Open Literature**

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

http://iac.dtic.mil/dacs

Access the DACS ROI Dashboard!

http://www.thedacs.com/databases/roi/
About the Software Tech News

STN Editorial Board

Philip King
Editor
ITT Industries, DACS

Paul Engelhart
DACS COTR
Air Force Research Lab (IFED)

Morton A. Hirschberg
Editorial Board Chairman
Army Research Lab (retired)

Ellen Walker
ITT Industries, DACS

Thomas McGibbon
DACS Director
ITT Industries, DACS

About This Publication:
The Software Tech News is published quarterly by the Data & Analysis Center for Software (DACS). The DACS is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC). The DACS is technically managed by Air Force Research Laboratory, Rome, NY and operated by ITT Industries, Advanced Engineering and Sciences Division.

Article Reproduction:
Images and information presented in these articles may be reproduced as long as the following message is noted:

“This article was originally printed in the Software Tech News, Vol. 9, No. 1. Requests for copies of the referenced newsletter may be submitted to the following address:

Philip King, Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil


In addition to this print message, we ask that you send us three copies of any document that references any article appearing in the Software Tech News.

Cover Design by Joseph Barbaccia,
ITT Industries

To Subscribe to this Publication Contact:

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil
Web: www.dacs.dtic.mil

Distribution Statement:
Unclassified and Unlimited
In This Issue

Tech Views ........................................................................................................1

The Measurement Challenge of High Maturity .................................................4

Estimating System-of-Systems Development Effort ...........................................13

Measurements: Managing a Large Complex Modernization Effort —While Protecting Your Program from the “Katrina Factor” .................................17

Building Systems Using Software Components ...............................................20

Advertisement

The DoD Software Tech News is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to the DoD Software Tech News in paper copy, the Tech News will also be placed on the Data & Analysis Center for Software’s website (http://iac.dtic.mil/dacs/), exposing your product, organization, or service to hundreds of thousands of additional eyes.

Interested in learning more? For rates, layout information, and requirements contact:

Philip King, STN Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400
Phone: (800) 214-7921
Fax: (315) 334-4964
E-mail: news-editor@dacs.dtic.mil

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Return Service Requested