SERVICE LEVEL AGREEMENTS AS VEHICLES FOR MANAGING ACQUISITION OF SOFTWARE-INTENSIVE SYSTEMS

CDR LEONARD T. GAINES, USN AND
DR. JAMES BRET MICHAEL

Service level agreements (SLAs) can be used as a means to manage the acquisition of software-intensive systems. The SLAs support performance-based acquisition by stating in measurable terms the service to be performed, the level of service that is acceptable, the way in which the service level is to be measured, and the incentives for the provider of information technology products and services to meet the agreed-to target levels of quality. The SLAs are traditionally used in outsourcing contracts for post-production support. This article proposes a new approach by using SLAs in software acquisition to support quality and process control throughout the entire lifecycle of a software-intensive system. This article defines SLAs, discusses software quality, and describes how SLAs can be utilized to incorporate requirements pertaining to product, process, project, and deployment quality throughout the software lifecycle.

As advances in information technology (IT) increase worker productivity and encourage the adoption of new ways of conducting business, organizations and end users become ever increasingly dependent upon that technology. Consequently, the strategic and tactical advantages that IT affords an organization places pressure on the organization's IT department to provide quality services and products. Interruptions to software-intensive systems are having a far greater impact than before in terms of opportunity loss, revenue loss, customer dissatisfaction, and inefficiency. As managers realize that their mission-critical processes are tied to IT services and products, they are demanding correspondingly greater levels of both performance and quality in these services and products, despite the fact that acquiring and maintaining software-intensive systems is increasingly challenging from both a managerial and technical perspective.

In addition, an organization may not have enough personnel with the right mix of IT skills, knowledge, and experience within their organization to provide high-enough quality IT products and services for the organization's employees and business partners. Rather than hire IT specialists or invest in training for its internal staff, the organization may have the option to outsource to external service providers (ESPs) their IT services and products. This strategy has been followed by the U.S. Department of Defense (DoD) in procuring large software-intensive systems such as the well-known Navy/Marine Corps Intranet (NMCI) and Ballistic Missile Defense System (BMDS). The hope is that ESPs can readily supply different mixes of IT specialists to meet the ever-changing requirements an organization has for IT products and services, at a lower cost than enlarging, shrinking, or retraining its internal IT staff. In addition to providing access to cutting-edge technology and skilled staff, ESPs share the project risk and make it possible for organizations to concentrate on core competencies (King, 2001).

Reliance on outsourcing adds another layer of complexity onto the management of information systems.

Reliance on outsourcing adds another layer of complexity onto the management of information systems. Outsourcing efforts require additional discipline and management oversight that may not be necessary with in-house development or maintenance of information systems. Outsourcing requires skill in software acquisition as well as project management. Program managers need to be accomplished in software-requirements engineering, software development, the contracting process, requirement-change management, contract management and oversight, and contractor-relationship management.

When outsourcing software-intensive programs, the program manager must be explicit in stating the services to be performed, in addition to identifying the metrics and means to determine whether the contractor has satisfied those requirements. However, it is challenging to develop, manage, and enforce a contract for software services that will ensure that the contractor delivers the specified services or end product within prescribed levels of performance and quality.

Managing software-intensive information systems can also be challenging. Utilizing the latest technology to exploit information requires highly developed intellectual and managerial skills. The difficulty in managing these systems has been demonstrated by the numerous system development and maintenance projects within the U.S. Department of Defense. Many of these projects lacked sound planning, had inadequate controls, were without effective measurements for success, and failed to meet user expectations (U.S. General Accounting Office [GAO], 2001; Department of Defense, Office of the Inspector General [DoD OIG], 2000). However, the private sector also has problems managing IT projects. The 2003 Standish Group's CHAOS research report indicates that of the 13,522 IT projects studied; only 34 percent of projects were considered a success, and only 52 percent of required features were incorporated into the released product (Standish Group, 2003).

Despite software's increased importance to organizations, the quality of software is often lacking.

Despite software's increased importance to organizations, the quality of software is often lacking. Some would argue, such as Mann (2002), that despite the advances in software engineering theory, processes, methodology, and tools, software quality is actually getting worse. Mann's reasons center around four main points: 1) developers are rushing software to market, 2) software is poorly designed, 3) developers and testers lack the requisite skills, and 4) programs are not managed well. Given the difficulties in developing software-intensive systems, performance and quality requirements should be well defined and monitored throughout the software's lifecycle. In this paper we explain how service level agreements (SLAs) can be utilized in software acquisition to improve the quality and management of software throughout its lifecycle.

SLAs are a means of incorporating Performance-based Service Contracting (PBSC) into software acquisition. SLAs are similar to a Quality Assurance Plan (QAP) in that they define quality and performance requirements, they contain penalties for non-performance, and they establish quality control methods to ensure compliance.

METHODS

In this article we identify areas throughout the lifecycle (requirements generation through post-production support) of software-intensive systems in which SLAs can be used to manage the contractual provisions of insourced or outsourced

IT services and products. In this section we define terms associated with SLAs and quality, in addition to describing how SLAs can be utilized in requirements engineering, design, post-production support, and program management to achieve target levels of performance and quality for IT services and products.

An SLA is a contractual mechanism between a provider of services and a customer that defines a level of performance (Sturm, Morris, & Jander, 2000). This agreement defines in measurable terms the service to be performed, the level of service that is acceptable, the means to determine if the service is being provided at the agreed upon levels, and incentives for meeting agreed upon service levels. SLAs contain quality-of-service (QoS) requirements, including how QoS is to be measured. SLAs also set forth the roles and responsibilities—in contractual form—of both the organization and the provider of the services and products.

...Ever-growing reliance on software-intensive systems for conducting business and recent advances in both techniques and tools for specifying SLAs and monitoring compliance to SLAs have all contributed to the rise in the acceptance of SLAs within the public and private sectors.

SLAs are not a new concept: they have been around since the 1960s. However, ever-growing reliance on software-intensive systems for conducting business and recent advances in both techniques and tools for specifying SLAs and monitoring compliance to SLAs have all contributed to the rise in the acceptance of SLAs within the public and private sectors. If SLAs are not included in a system-acquisition contract, the acquiring organization has little leverage over the provider of the IT services or products to ensure that the organization's performance and QoS requirements are met.

There is a subtle distinction between an SLA and a traditional contract requirement. SLAs are also requirements, but they are different contractually: SLAs contain more detail, describe incentives for meeting thresholds, and specify incentives for meeting perfomance and QoS requirements. When SLAs are used, the contracting officer can withhold incentive payments or levy penalties if quality levels are not met during the time period stated in the SLA (usually a month or a quarter). Requirements are generally only measured at the end of a milestone decision. In most contracts, the only recourse if a requirement is not met is to cancel the contract, terminate any ongoing contractor support, or give the contractor a poor performance rating. Terminating a project can be difficult, especially if the software-intensive system to be acquired is mission-critical. SLAs specify the degree of recourse if a requirement is not met.

One of the ways that SLAs can help improve software quality is that they focus the attention of the acquisition professional on the nonfunctional requirements (e.g., reliability, security, testability) that otherwise might be ignored until the test phase in the absence of SLAs (Bass, Clements, & Kazman, 1998). SLAs contractually mandate the performance and quality attributes that users, program managers, and software engineers consider essential for a system to support the underlying business process. They help make explicit many of the quality factors that users may implicitly assume. SLAs also specify the quality metrics by which the software quality requirements are measured. Measuring and monitoring performance and quality of services and products makes it possible for an organization to judge whether performance and quality requirements have been met. Measurements also support early detection and resolution of problems with quality.

Quality can be viewed from numerous perspectives, and certain attributes are more preferable to others depending on the mission the software-intensive system is supposed to support.

Before discussing SLAs and how they can improve the performance and quality in the various phases of a software-intensive system's lifecycle, we first define the term *software quality*. There are numerous definitions of quality. The International Standards Organization (ISO) standard 9126 defines software quality as the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs. Another definition is that software quality is conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software (Pressman, 2001). Others believe that an IT system will not be perceived as a quality product if the product does not perform according to the end-user's perspective (Wiegers, 2002). The Institute of Electrical and Electronics Engineers (IEEE) standard 610-1990 does incorporate user needs: it defines software quality as the degree to which a system, component, or process meets specified requirements and meets customer or user needs and expectations (Schmidt, 2000).

In this article we adopt the IEEE definition and specialize it to encompass four specific areas of focus: 1) product quality, 2) project quality, 3) process quality, and 4) post-production quality. Each of the areas has quality attributes associated with it that describe the degree to which the software possesses certain characteristics. SLAs utilize quality metrics and threshold levels to assign quantitative measurements to the quality attributes. The quality metrics are then monitored to ensure compliance.

Quality can be viewed from numerous perspectives, and certain attributes are more preferable to others depending on the mission the software-intensive system is supposed to support. Numerous quality attributes and quality models have been developed to measure or predict software quality. Selecting the appropriate quality attributes for the project is the responsibility of the SLA development team, the program manager, and stakeholders.

PRODUCT QUALITY

Product quality is concerned with the relationships between the characteristics and attributes of a product and its ability to satisfy stated requirements and implicit needs; this area can also be referred to as end-product quality. In addition to the functional aspects of a system, the end users want the product to exhibit certain quality characteristics that will assist them in performing their task. From a user's perspective, some of the common quality attributes include availability, usability, integrity, interoperability, and reliability. Personnel involved in the development of software or its maintenance may be more concemed with the software attributes such as portability, testability, maintainability, and reusability (Wiegers, 2002).

The ISO standard 9126-1 quality model incorporates the following quality factors: functionality, reliability, usability, efficiency, maintainability, and portability.

There are numerous quality models that can be incorporated into SLAs that assign quantitative measurements to various product quality attributes. Some of the better-known quality models include those proposed by McCabe in which software complexity is a function of the number of conditional statements in the code, and those of Halstead in which software complexity is a function of the number of operators and operands (Ogasawara, Yamada, & Kojo, 1996). The ISO standard 9126-1 quality model incorporates the following quality factors: functionality, reliability, usability, efficiency, maintainability, and portability (Cross, 2002). There are also numerous software quality models that concentrate on specific quality factors such as complexity. Zuse (1991) identifies over 90 models for describing software complexity. Other quality models are specific to object-oriented systems (Coppick & Cheatham, 1992), specific languages (Pritchett, 2001), commercial off-the-shelf (COTS) components (Bertoa & Vallecillo, 2002), and others, are only applicable at run-time (Bass et al., 1998).

PROJECT QUALITY

Project quality is concemed with metrics that allow an organization to manage, track, and improve the quality of the software development effort; this area could be viewed as in-process quality management (Hilburn & Townhidnejad, 2000).

Some of the common project quality metrics that Motorola used to measure its software development projects included software defect density, adherence to schedule, estimation accuracy, reliability, requirements tracking, and fault-type tracking (Daskalantonakis, 1992). Another metric used in assessing project quality is risk, which can be defined as any variable within a project that results in project failure. General risk areas are schedule risk, requirements risk, budget risk, and personnel risk (Padayachee, 2002). There are a number of risk-assessment models including Gilb's (1993) risk heuristics, Boehm's (1991) classification of risk, Noguiera de Leon's (2000) risk assessment model, Keil's identification of risk factors (Keil, Cule, Lyytinen, & Schmidt, 1998), and risk models associated with enterprise software projects (Sumner, 2000).

PROCESS QUALITY

Process quality is concerned with the processes, planning, and controls used to develop and manage the software product. It could be argued that the quality of the development process is the best predictor of software product quality (Fenton, Krause, & Neil, 2002). Repeatable software processes such as the Software Engineering Institute's Capability Maturity Model for software (SW-CMM also known as CMM), which lists five levels of organizational maturity, and the ISO 9001, are designed to improve software quality, productivity, predictability, and time to market (McGuire, 1996). There is empirical evidence that supports the relationship between process maturity and software quality (Harter & Slaughter, 2000).

Process quality is concerned with the processes, planning, and controls used to develop and manage the software product.

Other models of process quality include the Software Engineering Institute's new Capability Maturity Model Integration (CMMI). CMMI integrates three CMM models into one to eliminate problems with different architecture, semantics, and approaches. Humphery (1996) also developed a process model called the Personal Software Process (PSP) to assist software engineers in producing quality software. Other process models include cleanroom engineering that has shown reduced errors per KLOC for small projects (Fenton & Neil, 1999), and the quality management metric (QMM) (Osmundson, Michael, Machniak, & Grossman, 2003). There are also numerous IEEE and ISO standards that provide processes on everything from software engineering product evaluation (e.g., ISO/IEC 14598) to selecting appropriate quality metrics (e.g., IEEE Std. 1061-1998).

POST-PRODUCTION QUALITY

The last area of focus is on post-production quality or deployed-application management. Many of the quality models involving deployed applications are concerned with software maintenance and the quality factors that make maintenance cost-effective. Some of the maintenance quality factors deal with ease of change (Royce, 1990), architectural design to promote maintenance (Garlan, 2000), defect management (Kajko-Mattsson, 1998), organizational structure (Briand, Melo, Seaman, & Basili, 1995), and change management (Bennett & Rajlich, 2000).

Quality does not only apply to the application itself, it is also concerned with the IT system as a whole, across distributed components. Part of that distributed system is the network. There are numerous quality metrics that can be applied to network QoS (Moeller et al., 2000). Quality metrics are also applied to the host server. Quality metrics such as application resource utilization (Aries, Banerjee, Brittan, Dillon, Kowalik, & Lixvar, 2002), bandwidth utilization (Eager, Vernon, & Zahorjan, 2001), concurrent user management (Aweya, Ouellette, Montuno, Doray, & Felske, 2002), and server performance (Gama, Meira, Carvalho, Guedes, & Almeida, 2001) are also utilized to address system-level quality. Other areas where quality metrics are applied include total cost of operation (TCO), help desk support metrics, backups, storage, configuration management, and security.

When developing the SLAs, it is essential that all stakeholders be represented in this effort, as the system requirements need to represent all of their needs for quality.

There are numerous software quality models and metrics that can be incorporated into SLAs. The models or quality factors chosen will depend upon those quality attributes that best support the underlying business processes of the organization acquiring the software-intensive system. SLAs should only incorporate software metrics that are meaningful, quantitative, and measurable.

When developing the SLAs, it is essential that all stakeholders be represented in this effort, as the system requirements need to represent all of their needs for quality. The development team, consisting of representation for all stakeholders, must resolve a number of issues. These include, for instance, identifying quality requirements, determining the various levels of service that are needed, detailing quality metrics that are meaningful and measurable, assessing risk, resolving conflicting quality requirements, and prioritizing the quality requirements. When quotations are received from vendors (i.e., the providers of IT services and products), the group also needs to perform a cost-benefit analysis based on the levels of quality thresholds desired and the budget allotted for the acquisition. The group development of SLAs helps the various stakeholders understand each others' biases, viewpoints, concerns, terminology, and perceptions—that understanding is essential in requirements engineering.

REQUIREMENTS ENGINEERING

Requirements engineering provides the building blocks for all other efforts in the software-engineering process, so if quality is not addressed in the initial requirements analysis, it is usually addressed at the end of the project in the form of testing. If quality requirements are implemented too late, the architecture that was already developed will dictate the solution space for addressing quality problems that are discovered, or the acquisition authority will need to approve major contract modifications to permit changes to the requirements, architecture, and other high-level system artifacts.

Requirements engineering provides the building blocks for all other efforts in the software-engineering process, so if quality is not addressed in the initial requirements analysis, it is usually addressed at the end of the project in the form of testing.

The process of developing the SLAs is part of the larger requirement-engineering process and fosters discussion of the following: the goals of the system that must be met, the processes and tasks that the system must perform to meet those goals, and any operational and organizational needs, policies, standards, and constraints. Discussions necessary to develop the SLAs will generate information about the application domain, business and organization processes/culture, and the intended operating environment that the system will be placed in. The discussions will help the requirements engineer capture tacit knowledge, identify constraints, prioritize requirements, and justify how quality factors support business needs. Additionally, SLAs require quantifiable quality metrics, so those quality requirements that cannot be measured, do not provide value, or are not realistic should not be accepted by the SLA development team or the requirements engineer.

In many organizations, senior management's involvement in requirements engineering can be low (Bubenko, 1995), resulting in requirements that may not be related to business visions and objectives. SLAs help mitigate this problem because they define specific performance parameters that are tied to business processes. As such, every SLA performance requirement must be analyzed, and validated to ensure that it is meaningful, cost-effective, and that they add to improving overall performance. Senior management may also take more notice when there are contractual repercussions (i.e., penalties or rewards) associated with the requirements.

Incorporating standards in SLAs provides a common methodology that makes management easier for program managers and contracting officers as they provide the basis against which activities can be measured and evaluated (Horch, 1996).

By defining meaningful and measurable metrics in the SLAs, the end-users, business managers and software engineers have realistic quantifiable requirements that can be used to develop architectures, designs, and other software artifacts.

DEVELOPMENT

The real contribution of SLAs in obtaining quality software is that the quality factors addressed in the SLAs drive significant architectural and design decisions. If developers know which of the characteristics are most critical to project success, they can select the architecture, design, programming, and verification and validation approaches that best achieve the specified quality goals. When customer requirements have been collected and specified, architecting and designing translates the requirements into a blueprint that programmers can use to build the product. That design can be evaluated and monitored to ensure quality factors are adequately addressed. In this way, quality is designed in the beginning phases of the lifecycle where it is most cost-effective to do so.

The architecture of a software system models its structure and behavior (Shaw & Garlan, 1996). Architecture also shows the correspondence between the system requirements and the elements of the constructed system; so by analyzing the architecture it is possible to predict the quality of a product before it has been built. There are a number of methods to analyze architectures (Bass et al., 1998). However, these methods do not produce quantifiable quality measurements (Dobrica &

Niemela, 2002), although they can provide an estimate of how well the design will satisfy a particular quality factor.

SLAs also improve software quality in the development phase by contractually mandating that certain quality-control measures (e.g., adhering to specified standards and processes) be performed. There are numerous industry-approved standards that can be incorporated into SLAs. Incorporating standards in SLAs provides a common methodology that makes management easier for program managers and contracting officers as they provide the basis against which activities can be measured and evaluated (Horch, 1996). Standards can be applied to all aspects of development, maintenance, and operation of an information system.

Development processes can be specified in an SLA. Specifying specific processes has many of the same advantages of specifying compliance to standards. Applying well defined, standardized software-development processes can increase software quality and make the development effort more cost effective and predictable (Gnatz, Marschall, Popp, Rausch, & Schwerin, 2003). Specifying the processes in the SLA helps to ensure that they are recognized and adhered to.

Applying well defined, standardized software-development processes can increase software quality and make the development effort more cost effective and predictable (Gnatz, Marscahll, Popp, Rausch, & Schwerin, 2003).

SLAs also assist the testing effort by identifying business-critical processes, defining quantitative metrics to measure quality factors, identifying testing procedures, and ensuring testing is conducted throughout the system's lifecycle. SLAs can ensure that other audits such as documentation reviews, requirements reviews, design reviews, test plan reviews, user documentation reviews, and implementation reviews are conducted (Horch, 1996).

PROGRAM MANAGEMENT

Program managers have to ensure that quality considerations are addressed early in the lifecycle and they must provide the proper amount of oversight to ensure those quality factors are incorporated into the final product. SLAs can assist program managers in many of the tasks necessary to ensure quality is delivered in the final and future versions of a product.

In order to reduce and manage risk, program managers need to measure or monitor contractor, project, and system performance throughout the project's lifecycle. This will help ensure that standards, processes, and quality requirements are being met. SLAs mandate monitoring of the quality requirements associated with product, process, project, and post-production quality. If quality levels are not met, program managers and the contractor are informed of the violation and potential risks; that knowledge may lead to closer monitoring or corrective action to reduce risk and improve quality.

The development of SLAs provides information that will assist the program manager in managing the project's finances. SLAs help financial management in determining the scope of the project (e.g., determining areas of responsibilities in an end-to-end performance SLA), identifying business-critical processes and functions, they help to allocate costs among business units, they provide justification for service-related expenditures, and they help coordinate the IT strategy with business strategies (e.g., applying funds toward services to support critical processes).

Contract oversight is easier when SLAs are established because they can be used to define the levels of service expected, explain how the measurement will be conducted, identify responsibilities, and set forth escalation procedures.

Quality control consists of the actions necessary to certify that desired standards, processes, and quality requirements are adhered to during design, implementation, and production (Tricker & Sherring-Lucas, 2001). SLAs are a quality-control mechanism. SLAs help the program manager by contractually mandating that certain quality-control methods be implemented.

Contract oversight is easier when SLAs are established because they can be used to define the levels of service expected, explain how the measurement will be conducted, identify responsibilities, and set forth escalation procedures. The monitoring mandated by SLAs helps the program manager determine whether the contractor is meeting the quality requirements. When determining what quality metrics are included in the SLAs, it is helpful to determine the behavior you want from the contractor, and determine what measurements will most likely encourage that behavior (Kendrick, 2003).

SLAs help institutionalize a change review board to continually review the SLAs, evaluate new requirements, and ensure maintenance actions do not affect the SLAs.

The change review board ensures that only authorized changes are enacted, that changes are tested against quality levels, that changes conform to architectural constraints, and those changes are properly documented. SLAs can also be written to specify quality requirements that deal specifically with the accuracy or effectiveness of configuration identification, configuration control, configuration accounting, and configuration audits.

In addition, SLAs provide a basis for common understanding of the services that will be performed, the levels of service to be expected, how they will be measured, as well as define the responsibilities of both parties to an SLA. Both parties must mutually agree upon contractual SLAs, or there will never be a contract. It is commonplace to negotiate on the services and the performance levels that are requested and ultimately agreed upon. An SLA should contain a definition of service requirement that is both achievable by the provider and affordable by the customer. The customer and the ESP must also define a mutually acceptable set of indicators of the quality of service (Sturm et al., 2000). Note that SLAs can and should be modified throughout the lifecycle of a system as requirements change, technology improves, and efficiencies are gained.

The use of SLAs helps to ensure that customer expectations are being met by establishing performance and quality objectives, and providing the metrics and a measurement methodology to ensure that those requirements are being met.

The use of SLAs helps to ensure that customer expectations are being met by establishing performance and quality objectives, and providing the metrics and a measurement methodology to ensure that those requirements are being met. An important part of customer service is monitoring the performance of the system to ensure that it is supporting the critical business processes in a manner acceptable to the customer. The fact that SLAs specify what is acceptable makes it easier for program managers to manage expectations.

SLAs can be valuable to organizations that lack the proper IT skills to contract with ESPs. Template SLAs, or pre-existing SLAs, which represent best business practices, can be used as a starting point in formulating SLAs that are unique to specific applications. SLA templates are a starting point for program managers to develop their own SLAs. The templates are helpful in generating questions regarding services and service levels that should be provided to support or develop applications. The program managers only have to modify the existing SLAs to best support their application.

DEPLOYMENT SUPPORT

Quality control does not stop once a software product has been deployed. Quality requirements still need to be applied to the application performance, maintenance efforts, and hosting services throughout its lifecycle. Monitoring the performance of the application once it is deployed is essential in quality control and maintaining customer satisfaction. Much of the application performance monitoring in the initial phases of deployment is to validate product-quality requirements identified in the initial requirements. However, in the deployment environment there is also an emphasis on monitoring system performance in terms of resource utilization, application security, application reliability, application maintainability, database performance, and server performance.

Quality requirements associated with deployed software take a holistic view of the entire system, including distributed components. Quality requirements should be applied to network performance, host environment security, disaster recovery, storage solutions, problem management, concurrent user management, as well as end-to-end performance. Monitoring of the network, hardware, operating system, and the application not only assists in problem resolution, but trend analysis can indicate potential problems before they become unwieldy.

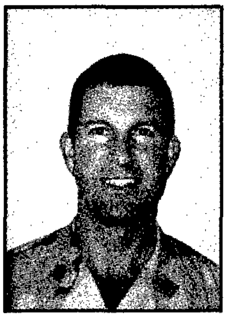
SLAs can be utilized for maintenance actions when patches or new versions need to be developed and deployed. SLAs utilized in the development phase may be used in some circumstances with maintenance actions.

CONCLUSION

SLAs are performance-based instruments for acquiring software. SLAs can be used to specify software quality, but they are also useful for specifying performance requirements. We have described how SLAs can drive product, process, project, and deployment quality solutions. SLAs can help ensure that quality requirements are identified and established early in the development cycle in order to be incorporated into preliminary designs. SLAs can help program managers establish quality controls to monitor and manage the various aspects of the projects. SLAs also carry sufficient weight through incentives to focus management and contractor attention on the quality issues for a software-intensive system that affect the ability of the acquiring organization to perform its mission.

Future research that can build upon the foundations set forth in this paper include evaluating SLAs in actual contracting to analyze quality improvements, determining which quality models and quality attributes are best suited for different types of IT systems, and evaluating or generating tools necessary for measuring end-to-end SLA measurements, such as response time and availability.

All organizations want world-class quality levels, but achieving those quality levels requires a holistic view of quality that incorporates leadership support, repeatable and measurable quality processes and controls, resource planning, vision, customer support, and service-level management. Organizations must do more than identify and incorporate quality attributes in their requirements: they must also monitor quality metrics to ensure those quality requirements are being met. SLAs are instruments that can be used to establish quality controls. Quality is not something that is inherent in the development process; it must be planned, monitored and incorporated as part of standard business practices.



CDR Leonard T. Gaines, USN is a Supply Corps Officer currently assigned to the Joint Logistics Operation Center at the Defense Logistics Agency. He has a master's degrees from the Naval Postgraduate School and the Industrial College of the Armed Forces. He has served on three ships and has had shore assignments in logistics, acquisition, and information technology. He is a member of the Acquisition Professional Community.
(E-mail address: gaines@dla.mil)

Dr. James Bret Michael is an Associate Professor of Computer Science and Electrical & Computer Engineering at the Naval Postgraduate School. His research and teaching interests center on distributed computing and software engineering as they apply to ballistic missile defense and information warfare. He serves on several government, editorial, and industry advisory boards. Michael has a doctorate degree from George Mason University and is a senior member of the Institute of Electical and Electronic Engineers (IEEE).
(E-mail address: bmichael@nps.edu)

AUTHOR **BIOGRAPHIES**

REFERENCES

Aries, J., Banerjee, S., Brittan, M., Dillon, E., Kowalik, J., & Lixvar, J. (2002, June). Capacity and performance analysis of distributed enterprise systems. *Communications of the Association of Computing Machinery* (ACM).

Aweya, J., Ouellette, M., Montuno, D., Doray, B., & Felske, K. (2002, January/ February). An adaptive load balancing scheme for web servers. *International Journal of Network Management, 12*, 3–39.

Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice.* Reading, WA: Addison-Wesley.

Bennett, K., & Rajlich, V. (2000, June). Software maintenance and evolution: A roadmap. *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, 73–87.

Bertoa, M., & Vallecillo, A. (2002, November). Quality attributes for COTS components. Retrieved October 12, 2004, from http://alarcos.inf-cr.uclm.es/ qaoose2002/docs /QAOOSE-Ber-Val.pdf

Boehm, B. (1991, January). Software risk management: Principles and practice. *IEEE Software, 91*, 32–41.

Briand, L., Melo, W., Seaman, C., & Basili, V. (1995, April). Characterizing and assessing a large-scale maintenance organization. *Proceedings of the Seventeenth International Conference on Software Engineering*, Seattle, WA, 133–143.

Bubenko, J. (1995, March). Challenges in requirements engineering. *Second IEEE Symposium on Requirements Engineering*, York, England, 160–162.

Coppick, C., & Cheatham, T. (1992, March). Software metrics for object-oriented systems. *Proceedings of the 1992 ACM Annual Conference on Communications*, Kansas City, MI, 317–322.

Cross, S. (2002, December). A quality doctrine for software: Do it right the first time. *Proceedings of the ninth Asia-Pacific Software Engineering Conference*, Gold Coast, Australia, 187–194.

Daskalantonakis, M. (1992, November). A practical view of software measurement and implementation experiences within Motorola. *IEEE Transactions on Software Engineering, 18*, 998–1010.

Department of Defense, Office of the Inspector General. (2000, July 13). *Summary of audits of acquisition of information technology.* (DoD OIG Report D-2000-162). Washington, DC: U.S. Government Printing Office.

Dobrica, L., & Niemela, E. (2002, July). A Survey on software architecture analysis measures. *IEEE Transactions on Software Engineering, 28*(7), 638–653.

Eager, D., Vemon, M., & Zahorjan, J. (2001, September/October). Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Knowledge and Data Engineering, 15*(5), 742–757.

Fenton, N., Krause, P., & Neil, M. (2002, July/August). Software measurement: Uncertainty and causal modeling. *IEEE Software*, 116–122.

Fenton, N., & Neil, M. (1999, September/October). A critique of software defect prediction models. *IEEE Transactions on Software Engineering, 25*(5), 675–689.

Gama, G., Meira, W., Carvalho, M., Guedes, D., & Almeida, V. (2001, November). Resource placement in distributed e-commerce servers. Proceedings of Globecom 2001, San Antonio, TX, 1677–1682.

Garlan, D. (2000, June). Software architecture: A roadmap. *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, 93–101.

Gilb, T. (1993). Risk management: A practical toolkit for identifying analyzing and coping with project risk. Retrieved October 12, 2004, from http://www.result-planning.com/Download/Risk.pdf

Gnatz, M., Marschall, F., Popp, G., Rausch, A., & Schwerin, W. (2003, June). The living software development process. *Software Quality Professional*, 4–16.

Harter, D., & Slaughter, S., (2000). Process maturity and software quality: A field study. *Proceedings of the Twenty-first International Conference on Information Systems*, Brisbane, Australia, 407–411.

Hilburn, T., & Townhidnejad, M. (2000, March). Software quality: A curriculum postscript. *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, Austin, TX, 167–171.

Horch, J. (1996). *Practical guide to software quality management.* Boston, MA: Artech House Publishers.

Humphrey, W. (1996, May). Using a defined and measured personal software process. *IEEE Software*, 77–88.

Kajko-Mattsson, M. (1998, April). A conceptual model of software maintenance. *Proceedings of the Twentieth International Conference on Software Engineering*, Kyoto, Japan, 422–425.

Keil, M., Cule, P., Lyytinen, K., & Schmidt, R. (1998, November). A framework for identifying software project risk. *Communications of the ACM*, 76–83.

Kendrick, T. (2003). *Identifying and managing project risk: Essential tools for failure-proofing your project.* New York: AMACOM.

King, W. (2001, February). Guest editorial developing a sourcing strategy for IS: A behavioral decision process and framework. *IEEE Transactions on Engineering Management, 48*(1), 15–24.

Mann, C. (2002, July/August). Why software is so bad. *MIT Technology Review*, 33–38.

Nogueira de Leon, J. (2000, September). *A formal model for risk assessment in software projects*. Unpublished doctoral dissertation, Naval Postgraduate School, Monterey, CA.

McGuire, E. (1996). Factors affecting the quality of software project management: An empirical study based on the capability maturity model. *Software Quality Journal*, *5*, 305–317.

Moeller, M., Hochstetler, S., Glasmacher, P., Jewan, P., Pathre, M., Polacheck, J., Sampath, V., & Ziller, H. (2000). *Tivoli Netview 6.01 and friends.* Austin, TX: IBM Redbook.

Ogasawara, H., Yamada, A., & Kojo, M. (1996, March). Experiences of software quality management using metrics through the life-cycle. *Proceedings of the Eighteenth International Conference on Software Engineering*, Berlin, Germany, 179–188.

Osmundson, J., Michael, J., Machniak, M., & Grossman, M. (2003, September). Quality management metrics for software development. *Information and Management*, *40*(8), 799–812.

Padayachee, K. (2002, September). An interpretive study of software risk management perspectives. *Proceedings of the Annual Research Conference of the South African Institute of Computer Science and Information Technologists on Enablement through Technology*, Port Elizabeth, South Africa, 118–127.

Pressman, R. (2001). *Software engineering a practitioner's approach* (5th ed.). New York: McGraw-Hill.

Pritchett, W. (2001, September). An object-oriented metrics suite for Ada 95. *Proceedings of the Annual ACM SIGAda International Conference on Ada*, Bloomington, MN, 117–126.

Royce, W. (1990, November). Pragmatic quality metrics for evolutionary software development. *Proceedings of the Conference on TRI-ADA '90*, Baltimore, MD, 551–565.

Schmidt, M. (2000). *Implementing the IEEE software engineering standards.* Indianapolis, IN: Sams.

Shaw, M., & Garlan, D. (1996). *Software architecture perspectives on an emerging discipline.* Upper Saddle River, NJ: Prentice Hall.

Standish Group. (2003). *Chaos.* In *The Standish Group report.* West Yarmouth, MA: The Standish Group.

Sturm, R., Morris, W., & Jander, M. (2000). *Foundations of service level management.* Indianapolis, IN: Sams.

Sumner, M. (2000, April). Risk factors in enterprisewide information management system projects. *Proceedings of the SIGCPR Conference on Computer Personnel Research*, Chicago, 180–187.

Tricker, R., & Sherring-Lucas, B. (2001). *ISO 9001:2000 in brief.* Oxford, MA: Butterworth Heinemann.

U.S. General Accounting Office. (2001). *Major management challenges and program risks: Department of Defense.* (U.S. GAO Report GAO 01-244). Washington, DC: U.S. Government Printing Office.

Wiegers, K. (2002). *Software requirements* (2nd ed.). Redmond, WA: Microsoft Press.

Zuse, H. (1991). *Software complexity—measures and methods.* Berlin, Germany: Walter De Gruyter & Co.

Your Retirement Goals
Are you saving enough?