Consortium for IT Software Quality

CISQ Recommendation Guide

Effective Software Quality Metrics for ADM Service Level Agreements

Contributors:
Dr. Bill Curtis – Executive Director, CISQ
Bill Dickenson, Strategy on the Web – Contributor, CISQ
Chris Kinsey, Strategy on the Web – Contributor, CISQ

Table of Contents

Introduction

Service Level Agreements (SLA) are an integral part of the Application Development and Maintenance (ADM) process. SLAs have been used to define the working relationship between a service provider and customer since the early days of IT outsourcing. Yet many of the contracts written, even in the last 5 years, use fundamentally the same time-based SLAs for ADM. For example, SLAs for responding to a high severity ticket or the turnaround on a work request are common. SLAs that contract around the quality of the actual code produced are rare in contrast. While many of the SLAs are appropriate for infrastructure, the SLAs for application software focus on relatively indirect measurements. Given the tight linkage between support costs, code quality, and subsequent risk to business, **this must change**.

The average application costs 20% of its development cost, year on year, to support. Increasing the quality of the application from "average" to "good" reduces support costs to 12%, and "excellent" code can cost as little as 3-5% of development cost to support. This savings of 8% to 17% more than justifies the tooling and approaches required to write good source code at the outset. It is time to step into a new set of SLAs that drive the needed objectives of lower risk and lower cost. This CISQ Recommendation Guide will explain how to add software quality metrics to a service level agreement to drive improved application ROI.

Why Use Measures from CISQ?

CISQ has developed international standards for software quality that are ideal for use in service level agreements for three reasons. First, CISQ quality characteristic measures are consistent with ISO/IEC 25000 definitions. ISO has defined, at the technical level, software quality characteristics for security, reliability, maintainability and other non-functional characteristics of software. CISQ has taken next steps to standardize the measurement of these attributes at the source code level. This allows software quality tools to automate the detection of critical violations of good architectural and coding practice related to each characteristic in the source code of software. Second, CISQ metrics are automatable. Automated measures make it possible to track software quality over time in a consistent, predictable, and cost-effective manner. Third, CISQ's parent organization, the Object Management Group (OMG), is an internationally recognized standards organization that is committed to producing software standards for the global IT industry.

SLA Approach

The goal of a SLA is to incent vendor performance to a defined standard. In recent years, the SLA has occasionally become a tool to penalize the vendor or generate customer budget windfalls. Suppliers have become adept at avoiding SLA penalties in those situations. The approach described in this guide uses CISQ software quality metrics to lower risk, drive support costs down, or both. The result is an SLA structure that is fair to the supplier and fair to the client.

The major focus of next generation SLAs is in the service levels which reflect some level of defect or violation of good coding principles. Also, there are violations which are so egregious that they rise to the level of an obligation. These are effectively "zero tolerance violations" that must be remediated before the code is put into QA. There are not many of these and the inclusion of these in the obligations table should not be difficult. The "zero tolerance violations" or obligations do represent major flaws in coding.

Characteristics of Application SLAs

SLAs should be based on clear metrics derived from readily available data. The metrics should be based on well-defined international standards so that both parties, supplier and client, share a common understanding of the results to be achieved. In addition, each SLA should tie to a goal of driving lower total cost of ownership and/or lower risk. Software quality metrics contracted in SLAs should have the following characteristics:

- Have a clear, unambiguous definition suitable for inclusion in a contract. CISQ is one good source. The metrics proposed for these SLAs are based primarily on work done by CISQ. CISQ standards are the result of extensive research and decades of experience.

- Have an industry "baseline" for each metric with enough experience to prove the benefits of achieving the baseline and creating a standard of legitimacy for the contract. Recognizing that any metric must be fair and compel the right behavior, picking metrics that have industry data relating the metric to risk or cost outcomes focuses the development process on actual benefits. Existing work should use a client-specific baseline which will be the starting point for change.

- Be collected automatically. Manual processes break down and are subject to interpretation, human error, or manipulation. Automated collection produces deterministic, consistent results. While there can be some debate on the "perfect" answer, automated collection produces the same result each time, is not influenced by outside factors, and is consistent across platforms and languages.

- Be adjusted for volume. The type of ADM work varies month to month. The number of defects introduced in a given month may be high as the result of a large introduction of functionality. Conversely, a small number of violations might be an issue if minimal functionality was added. Adjustment based on volume is essential for fairness and to avoid volume-related variation in measurement results.

With these dimensions in mind, many metrics become suitable for SLAs. The denominator is typically a measure of software size, such as Automated Function Points (AFP) standardized by CISQ, or other measures of software volume.

Basic SLA Structure

A large majority of the ADM contracts contain a structure where there is an "at risk" cap, typically a percentage (e.g., 10%) of the monthly contract value, and with a weighting for each SLA. The weighting factor represents the amount of the "at risk" amount that will be forfeited if the SLA is triggered.

Example: A $2M ADM contract with a 10% cap would have $200K "at risk." This $200K would be divided between the SLAs. Each SLA fault will cost a percentage of the $200K.

In the early days of SLA management, the SLA weights added to 100%. Limiting this to 100% meant that the supplier would have to miss all of the SLAs to pay the full at risk amount which rarely happened. Clients felt that faulting on multiple SLAs should trigger the entire cap.

In most modern contracts, the weighting factor for software quality metrics is typically "over weighted" at 200% to allow clients to put additional weight on the metrics that are most important to them. Allowing a total weight over 100% creates a situation where the supplier could fault on 2-3 SLAs, out of a possible 10, and pay the at risk amount. Overweighting can consume the at risk amount by missing as few as two metrics (e.g. Robustness and Security). The cap is the maximum, and no matter what the weighting, the at risk is the hard stop. In most contracts this is represented as 10/200, 10% at risk, 200% over weighted.

A recent trend in contracts has been to include a bonus percentage which can be used to either recoup the amount lost from a previous penalty, or to incent areas of importance and improvement. While typically applied to the entire contract, this can be assessed for each SLA, and is usually an end of budget cycle event. This tends to be 5%.

The time period for collecting and reporting SLAs can vary by contract. Monthly is most common but quarterly and annual periods may be more appropriate. A recent trend for new development is to analyze SLA compliance at the end of each bundle of defined activities, when performance of the entire work package can be measured. The results are applied to the cap for the same time period. No more than the Cap% of fees for the period will be at risk and any combination will still be subject to that cap.

Using a contract value of $2M, an at risk of 10/250%, and a bonus cap of 5%, a typical SLA contract table using the CISQ quality metrics might look like this. Note that each ADM contract is unique and the figures listed below are examples.

Name	Description	Type	Period	Baseline	Weight	Low	High	Annual Improvement
Security	The likelihood of potential security breaches of an application.	Unit	Monthly	0.02	35%	0.018	0.022	5%
		System	Monthly	0.02	35%	0.00	0.019	5%
Reliability	The risk of failure or defects that can result from changing an application.	Unit	Monthly	0.1	25%	0.09	0.11	5%
		System	Monthly	0.1	35%	0.07	0.10	5%
Performance Efficiency	How well the code handles unexpected events and how easily system performance can be reestablished.	Unit	Monthly	1	25%	0.9	1.1	2%
		System	Monthly	1	25%	0.8	1.0	2%
Maintainability	The difficulty and ease to maintain an application.	Unit	Monthly	3	25%	2.7	3.3	5%
		System	Monthly	3	25%	2.3	2.9	5%
				Total	200%			

The **Name** and **Description** are taken from the published CISQ standards. In order to use industry norms, these should be left as-is. While it is possible to simply "point" to the standard, to keep the contract "whole" the baseline, calculations, and source should be included in the contract. It is important to avoid loading every possible SLA into a contract. Focusing on quality, risk, and cost metrics will lead to more constructive results than including dozens of metrics. Having an excessive number of metrics (termed "herding") will simply lead to avoidance behavior on the part of the supplier and a dilution of the Cap%. Suppliers have been known to deliberately choose a penalty over the cost of performance.

Type refers to the classification of defects as unit level or system level defects. There are times when it makes sense to weight system level defects higher than unit level defects. Typically, system level defects are a greater risk to business.

In all of the cases described, the formula is Defects/AFP. If other formulas are used, they should be included as a separate column.

Period is the measurement period where the metrics are collected and penalties assessed. It is typically tied to the billing period. For new development projects with service levels, this is when the code is turned over for sign-off and represents the supplier's agreement that these levels can be assessed and will pass. Payment for new development should be withheld until the code achieves the necessary standards.

Baseline can be set from an industry repository such as ISBSG, CAST Appmarq, or an internal baseline derived from an agreed upon number of months of data which should factor in seasonal spikes common to the company or industry. The CAST Appmarq repository currently contains structural quality data on over 1800 applications consisting of over 2B lines of code and is the current standard for benchmarking structural quality across languages. If an internally generated baseline is to be used, the initial data must be collected and reported as they will be used for assessing SLA compliance. When collecting baseline data, the median should be used as the baseline number to avoid misleading results due to extreme values. Ideally, 6 months of data should be used if that includes any seasonality – such as seasonal fluctuations in the volume of changes implemented. Periodic baseline adjustments can be used over successive measurement periods from contract initiation as improvements are completed. For instance, in the case of maintaining poorly constructed code, the initial baseline may be set low recognizing the difficulty the service provider will have in working with the application. However, the baseline may be steadily raised over several measurement periods to represent improvements in the quality of the code the service provider is expected maintain as their maintenance work progresses.

While maintenance of existing work will likely use the baseline process, new development should follow the standards defined by one or more of the industry standards.

When a supplier has an interest in having a more aggressive baseline, the contract should be constructed to have milestone levels with a bonus paid out for achieving and sustaining those levels ahead of schedule. In any event, the baseline should improve Year-on-Year. If the initial baseline was unintentionally set low, the data from the first years achievement should be used as the basis for the next, increasing the year-on-year improvement.

The **trigger level** that causes a loss of the "at risk" amount is usually set either 10% below the industry average or the low value of the initial baseline. This represents a failure in service that warrants

immediate attention. Failing to achieve the threshold for 3 consecutive periods, or 3 failures within 5 consecutive periods, should trigger termination for cause. This does not mean that the contract is automatically terminated. Most termination for cause clauses are designed to bring the parties together for an immediate discussion as the client is not getting the service contracted for. As a practical matter, clients do not want to endure the switching costs of termination any more than vendors want to suffer the loss in revenue and reputation.

Reward Threshold is the level of achievement beyond which the supplier can earn back previous losses from failing to achieve thresholds or can be provided a bonus for exceeding expectations in a way that provides additional benefits to the customer. In the case of recouping losses, it represents a concerted effort by the supplier to remediate or improve the process. In the case of a bonus, the customer's objective is to receive additional benefits in the form of measurable cost or risk reductions.

Dead band is the range between the trigger level and the reward threshold. Service delivered within the dead band is considered acceptable and expected performance. The customer and service provider should hold periodic meetings to ensure that the measures, thresholds, and conditions of measurement are helping them achieve the customer's objectives.

Annual Improvement is different from adjusted thresholds in that it represents the expected amount of change to the SLA year-on-year rather than across successive measurement periods used to correct initial deficiencies in the code base. By default, continuous improvement tied to measureable and achievable thresholds should be built into application contracts. It certainly is a core principle behind CMM/CMMI, ITIL, and other best practice standards. If a supplier is meeting the SLAs, the baseline should be raised each year. If the supplier is exceeding the SLA, the baseline can be adjusted to a level appropriate for motivating continued improvement.

Setting SLA Measurements for Structural Quality Characteristics

CISQ recommends the following four OMG standard measures be applied to ADM contracts as part of a holistic measurement system: Security, Reliability, Performance Efficiency, and Maintainability. OMG's automated source code measures for Security, Reliability, Performance Efficiency, and Maintainability were developed by representatives from 24 CISQ member companies that included large IT organizations, software service providers, and software technology vendors. These measures were developed from lists of severe violations of architectural and coding practice known to cause problems. These structural quality measures can be normalized for the size of the application by using OMG's Automated Function Point or Automated Enhancement Function Point standards to produce density measures for use in SLAs.

Each of the four standard measures can be further divided into **unit** and **system** level violations. More advanced environments should consider weighting the system level violations more heavily than the unit level violations. Experience has shown that the system level defects tend to have higher risk than unit violations.

The following defect densities are listed for example purposes. Note that each application is unique and

setting a threshold for each quality measure must take into account agreed upon internal benchmarks or industry standards.

1) Security Violations per Automated Function Point
The MITRE Common Weakness Enumeration (CWE) database contains very clear guidance on unacceptable coding practices. In a perfect world, delivered code should not violate any of these practices. More realistically, all code developed for, or provided to a customer should have no violations of the Top 25 most dangerous and severe security violations, 22 of which are measureable in the source code and constitute in the OMG's Automated Source Code Security Measure. This measure is typically assigned a high weighting in SLA penalty calculations because of the financial and reputational damages that unauthorized intrusions and data theft can cause.

Security is not just an infrastructure layer issue as ample evidence in the press has shown. Vulnerability is most often found in the interactions between components which are often caused by inaccurate assumptions made by application developers about how other parts of the application work. Security is a system level issue in the code since user entry, validation, and data access is typically managed in different layers of a modern application. Many of the CWEs constituting this measure require overall system level analysis and measurement. This is one of the SLAs where it may be appropriate to have different standards based on an application suite. Online applications may have more of a zero tolerance policy.

Good Coding Practices Violations Unit Level (.02 / AFP)	**Good Architectural Practices Violations System Level (0.0 / AFP)**
• Use of hard-coded credentials • Buffer overflows • Missing initialization • Improper validation of array index • Improper locking • Uncontrolled format string	• Input validation • SQL injection • Cross-site scripting • Failure to use vetted libraries or frameworks • Secure architecture design compliance

2) Reliability Below 0.1 Violations per Automated Function Point
In any code there are data conditions that could cause the code to break in a way that allows an antagonist to gain access to the system. This can cause delivery failures in the expected functionality of the code. Reliability measures how well the code handles unexpected events and how easily system performance can be reestablished. Reliability can be measured as weaknesses in the code that can cause outages, data corruption, or unexpected behaviors. The Reliability metric has been operationalized in OMG's Automated Source Code Reliability Measure which is composed from 29 severe violations of good architectural and coding practice that can cause applications to behave unreliably.

Good Coding Practices Violations Unit Level (.1 / AFP)	**Good Architectural Practices Violations System Level (.09 / AFP)**

• Protecting state in multi-threaded environments • Safe use of inheritance and polymorphism • Resource bounds management, Complex code • Managing allocated resources, Timeouts	• Multi-layer design compliance • Software manages data integrity and consistency • Exception handling through transactions • Class architecture compliance

3) Performance Efficiency Below 1.0 Violations per Automated Function Point

Performance Efficiency measures how efficiently the application performs or uses resources such as processor or memory capacity. Performance Efficiency is measured as weaknesses in the code base that cause performance degradation or excessive processor or memory use. The Performance Efficiency metric has been operationalized in OMG's Automated Source Code Performance Efficiency Measure.

Good Coding Practices Violations Unit Level (1.0 / AFP)	**Good Architectural Practices Violations System Level (1.0 / AFP)**
• Compliance with Object-Oriented best practices • Compliance with SQL best practices • Expensive computations in loops • Static connections versus connection pools • Compliance with garbage collection best practices	• Appropriate interactions with expensive or remote resources • Data access performance and data management • Memory, network and disk space management • Centralized handling of client requests • Use of middle tier components vs. procedures/DB functions

4) Maintainability Violations Below 3.0 per Automated Function Point

As code becomes more complex, the change effort to adapt to evolving requirements also increases. Organizations that focus on Maintainability have a lower cost to operate, faster response to change, and a higher return on investment for operating costs. Code is rarely written and maintained by the same resources. In today's environment code may even have been originally developed by a different firm. It is important that code can be easily understood by different teams that inherit its maintenance. Up to 50% of maintenance effort is spent understanding the code before modification. Maintainable, easily changed code is more modular, more structured, less complex, and less interwoven with other system components, making it easier to understand and change. Maintainability measures how easily an application's code can be understood and how efficiently it can be changed or modified. The Maintainability metric has been operationalized in OMG's Automated Source Code Maintainability Measure which is composed from 20 severe violations of good architectural and coding practice that make code unnecessarily complex.

Good Coding Practices Violations	**Good Architectural Practices Violations**

Unit Level (3.0 / AFP)	System Level (2.8 / AFP)
• Unstructured and duplicated code • High cyclomatic complexity • Controlled level of dynamic coding • Over-parameterization of methods • Hard coding of literals • Excessive component size	• Duplicated business logic • Compliance with initial architecture design • Strict hierarchy of calling between architectural layers • Excessive horizontal layers • Excessive multi-tier fan-in/fan-out

Obligations

As described earlier in this guide, the supplier has an obligation to remediate certain violations as their presence in the code renders the code unusable. These violations represent misfeasance by the supplier. Depending on the application, the client may choose to add or remove items from this list based on their particular needs. Some of these violations may be acceptable for some applications, while a major issue for others. This set should be documented and agreed upon by the supplier as the penalty for violation is severe. The penalty should only apply to code created by the supplier, or code modified by the supplier.

Examples include:

- SQL injection: The SQL injection violation is so easily exploited that no such weaknesses will be tolerated in delivered code and must be removed at the service provider's expense before acceptance or operational deployment.
- Failing to include timeouts for threads: Threads without timeouts will continue to consume resources until the system fails.
- Failing to include/use indices in queries that act on tables over 10meg.
- Objects with more than an allowable number of instructions that may not be allowed in the delivered code.

Conclusion

By embedding these SLAs and Obligations into the ADM contract, the total cost of ownership will go down and cash will be freed up for new initiatives. The payoff as reported in recent publications indicates that the cost of increased tooling is typically recovered before the application moves into user acceptance testing. 52% of all defects are discovered not during testing but by the users in production - an unacceptable situation.

The goal of this new approach to SLA management is to focus on objective, repeatable, financially quantifiable software quality metrics which will drive behaviors beneficial to all stakeholders in the process. At the same time, the use of such metrics focuses on outcomes that are more appropriate to

return on investment discussions than the activities ADM SLAs have traditionally measured.

Virtually every other aspect of the business is managed with measures — cost, process efficiency, production waste, turnover, time to delivery, and return on capital, to name just a few. If ADM is to continue to improve, the results need to be measurable and support the goals of the business. Faster change, higher reliability, and lower cost are all very realistic outcomes. ADM needs to step up and be accountable for better business results. The measures provide leading indicators of operational results and therefore point the way.

Contributors:

Dr. Bill Curtis is the Executive Director of CISQ. He is best known for leading the development of the Capability Maturity Model (CMM) while at the Software Engineering Institute. CMM and its successor CMMI have become the global standard for evaluating the capability of software development organizations. He has 38 years' experience in software development including validating that software metrics predicted developer performance in General Electric Space Division, developing a corporate-wide software productivity and quality measurement system for ITT, leading research on advanced user interface technology and the software design process at MCC, co-founding TeraQuest the leading international consultancy on CMM-based improvement programs, and producing biennial global benchmarks on software structural quality (CRASH Reports) as SVP and Chief Scientist in CAST Software. In 2007 he was elected a Fellow of the IEEE for his career contributions to software process improvement and measurement.

Bill Dickenson is an independent consultant with Strategy On The Web, and former VP of Application Management Services for IBM, who brings decades of experience in application development, maintenance and integrated operations. Bill has had global responsibilities over the full range of IT from solution development, solution offerings, and leading global teams. He has spent a career working with C-level executives on business case development, services strategy, program and vendor management, business process reengineering, outsourcing engagements and implementation approaches. Bill has had professional experience with multiple industry verticals at Ciber, IBM, PwC, CSC, and DuPont in operations, new business development and executive leadership.

Chris Kinsey brings over three decades of operational, financial, and information technology experience spanning the gamut from entrepreneurial to global Fortune 100 C-suite experience. His skills have been utilized in high impact global environments at GE, PwC, and IBM. He has performed in business development, operational, and technical roles in both senior delivery and executive management. Most recently, Chris was the onsite client executive in several of IBM's largest outsourcing contracts. In this role he developed an expertise in contract metrics and governance.