Using Software Measurement in SLAs:

Integrating CISQ Size and Structural Quality Measures into Contractual Relationships

Contributors:
Dr. Bill Curtis – Founding Executive Director, CISQ
David Herron, David Consulting Group – Leader, CISQ Size Work Group
Jitendra Subramanyam, Hackett Group – Contributor, CISQ

Table of Contents

The Need for Standard Structural Measures of Software Applications

As more critical business functions within an organization are automated, IT is under increasing pressure to govern the quality of software received from suppliers, whether they are vendors, outsourcers, or system integrators while at the same time reducing cost. Better governance requires better measurement of application size, stability and quality in all of its manifestations—functional, non-functional or structural, and behavioral. These measures are being incorporated into contracts as the equivalent of Service Level Agreements (SLAs), targets that suppliers must achieve to avoid penalties. In some cases these SLAs involve productivity targets measured by the amount of business functionality delivered compared to the effort expended. In other cases they involve targets for the structural attributes of an application such as security or maintainability. Currently there are no industry standard definitions for automated measures of functional size and structural quality. This forces IT customers to define their own measures and leaves suppliers struggling with inconsistent definitions across their customer base.

Currently the functional size of an application is computed manually, making it both expensive to calculate and subject to inconsistent results from different counters. The most popular measure of functional size is Function Points as defined by the International Function Point User Group (IFPUG). Since this standard was defined for use by manual counters, there are vagaries in its definition that allow manual counters to use their judgment while counting. Consequently manual counts of the same software by two certified counters can differ by 10% or more. The business requires more. The business demands reduced cost, improved reliability and the flexibility to adapt to future change.

Structural quality is the extent to which an application's architecture and its implementation in source code is consistent with software engineering principles and best practices. Poor structural quality causes many of the high-impact business disruptions such as application outages, security breaches, corrupted data, and performance degradation. Industry data also demonstrate that poor structural quality creates enormous waste in the form of rework—30% to 50% of the effort in most organizations. The business agility of an enterprise is severely limited by the difficulty and time required to make changes to excessively complex and poorly constructed applications. Consequently, poor structural quality dramatically increases the cost and business risk of an application. Amazingly, as critical as structural quality is to the business there are no standard measures of these software quality characteristics.

Automation is necessary to reduce the cost and increase the accuracy of measuring both functional size and structural quality. Since modern business applications are an amalgamation of languages and technology platforms, measuring size and quality at the component level alone is not enough. The most devastating structural flaws frequently involve unanticipated or misunderstood interactions between several layers of the application, and such flaws can take 8-10 times longer to fix because of their complexity. Consequently, structural quality must be assessed by analyzing architectural characteristics across the various layers of an application as well the structural integrity of individual components.

This paper outlines how CISQ has defined size and quality characteristic measures.

The CISQ Size and Quality Characteristic Measures

The Consortium for IT Software Quality, now titled 'Consortium for Information & Software Quality,' was formed by the Software Engineering Institute (SEI) at Carnegie Mellon University (www.sei.cmu.edu) and the Object Management Group (www.omg.org) to develop standards for automating measures of functional size and structural quality so that they can be obtained quickly, inexpensively, and consistently. The automatable measures that have been defined by CISQ were selected by executives from CISQ's member companies.

CISQ members identified critical needs for a measure of functional size and four quality characteristics—Reliability, Security, Performance Efficiency, and Maintainability. These quality characteristics have been defined to be consistent with their definitions in the ISO 25010 Standard, which supersedes the earlier quality standard ISO 9126-3. However, the ISO 25000 series currently only provides conceptual definitions of these quality characteristics, and they are not sufficient to support automated measurement. CISQ fills this important void by defining the measures at a level required to support automation.

The specifications for the four CISQ Quality Characteristic measures are the result of work by CISQ members who listed violations of good architectural or coding practice related to each selected Quality Characteristic.

- Each of the CISQ Quality Characteristic measures is computed as counts of the most damaging violations of good architectural or coding practice within the context of a specific CISQ Quality Characteristic.
- Each violation represents a serious breach of a quality rule that is critical to a specific CISQ Quality Characteristic.
- Each of the CISQ Quality Characteristic measures consists of a count of severe violations that can be detected through automated software analysis of an application's source code.
- Quality rules can have language or platform-specific variations as necessary.
- Some violations can be marked as 'unacceptable violations' that must not appear in delivered code which are captured in contractual "obligations."

1. CISQ Automated Function Points

CISQ Automated Function Points have been defined to match as closely as possible with the IFPUG Function Point counting guidelines, while at the same time resolving any ambiguities so that the measure can be automated. Consequently, CISQ Automated Function Point counts can differ from those of an IFPUG-certified manual counter. Nevertheless, experience has shown that differences between manual and automated counts of Function Points occur on issues in the application where the greatest variance occurs among manual counters. The advantage of calculating CISQ Automated Function Points is their low cost, consistency, and speed.

Historical data on the relationship between functional size and effort provides a basis for estimating costs from functional specifications. CISQ Automated Function Points improve estimating by providing data that is more consistent and less expensive than manual counting. Further, CISQ Automated Function Points provide a more consistent basis for evaluating development or maintenance productivity and the factors that affect it. Counting CISQ Automated Function Points requires analysis through all the layers of the application to create a functional mapping of the application that can be split into data entities and user transactions. If calibration is required, such as when manual counts were used for estimates, this functional mapping will be used as the starting point of the calibration of any differences between CISQ Automated Function Point counts and those developed by the manual counter. Assessing the functional size of an application requires detection and analysis of the following elements:

- **Data Functions**
  - Internal logical files
  - External interface files

- **Transactional Functions**
  - External inputs
  - External outputs and inquiries

2. Reliability

Reliability measures the risk of potential application failures and the stability of an application when confronted with unexpected conditions. According to ISO/IEC/IEEE 24765, Reliability is the degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time. The reason for checking and monitoring Reliability is to reduce and prevent application downtime, application outages and errors that directly affect users, and enhance the company's business performance. Since modern applications integrate multiple components developed in different technologies, assessing application Reliability requires checks of the Reliability of each component as well as on the soundness of their structural integration into the larger application. Also most modern applications have adopted a layered approach where runtime issues have to be handled as close as possible to the place they occurred, minimizing their propagation to the end user and improving reliability. Thus, assessing the Reliability of an application requires at a minimum assessing the following software architectural and coding practices:

- **Architecture Practices**
  - Multi-layer design compliance
  - Software manages data integrity and consistency
  - Exception handling through transactions
  - Class architecture compliance

- **Coding Practices**
  - Protecting state in multi-threaded environments
  - Safe use of inheritance and polymorphism
  - Patterns that lead to unexpected behaviors
  - Resource bounds management
  - Managing allocated resources
  - Timeouts
  - Built-in remote addresses
  - Complex code

3. Performance Efficiency

Performance Efficiency assesses characteristics that affect an application's response behavior and use of resources under stated conditions (ISO/IEC 25010). The analysis of an application's Performance Efficiency attributes indicates risks regarding customer satisfaction, workforce productivity, or application scalability due to response-time degradation. It can also indicate current or projected future inefficient use of processing or storage resources based on wasteful violations of good architectural or coding practice. The Performance Efficiency of an application lies in each individual component's performance, as well as in the effect of each component on the behavior of the chain of components comprising a transaction in which it participates. Assessing Performance Efficiency requires checking at least the following software architectural and coding practices:

- **Architecture Practices**
  - Appropriate interactions with expensive and/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 versus stored procedures and database functions

- **Coding Practices**
  - 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

4. Security

Security assesses the degree to which an application protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization (ISO 25010). Security measures the risk of potential security breaches due to poor coding and architectural practices. Security problems have been studied extensively by the Software Assurance community and have been codified in the Common Weakness Enumeration (cwe.mitre.org). Assessing Security requires at least checking the following software architectural and coding practices that are included in the CWE Top 25 Security violations:

- **Architecture Practices**
  - Input validation
  - SQL injection
  - Cross-site scripting
  - Failure to use vetted libraries or frameworks
  - Secure architecture design compliance

- **Coding Practices**
  - Error and exception handling
  - Use of hard-coded credentials
  - Buffer overflows
  - Broken or risky cryptographic algorithms
  - Improper validation of array index
  - Missing initialization
  - References to released resources
  - Improper locking
  - Uncontrolled format string

5. Maintainability

Maintainability represents the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers (ISO 25010). Maintainability incorporates such concepts as changeability, modularity, understandability, testability, and reusability. Measuring maintainability is important for business or mission-critical applications where an organization must respond rapidly to customer or competitor-driven changes. It is also a key to keeping IT costs under control. The Maintainability of an application is a combination of compliance with best practices for coding that can be checked independently of the overall application context, the homogeneity with which coding rules are applied throughout the whole

application, and compliance with the core concepts of the chosen architecture and associated technologies. Assessing maintainability requires checking the following software architectural and coding practices:

- **Architecture Practices**
  - Strict hierarchy of calling between architectural layers
  - Excessive horizontal layers

- **Coding Practices**
  - Compliance with Object-Oriented best practices
  - Unstructured code
  - Duplicated code
  - Tightly coupled modules
  - Controlled level of dynamic coding
  - Cyclomatic complexity
  - Over-parameterization of methods
  - Encapsulated data access
  - Commented out instructions
  - Hard coding of literals
  - Excessive component size

Best Practices for Using CISQ Measures in SLAs

As with any other discipline, measuring Application Development and Maintenance (ADM) performance and software quality requires setting up achievable objectives, tracking levels of achievement (both general and specific), and ensuring positive motivation to achieve the best possible results. The purpose of incorporating CISQ Structural Quality metrics into SLAs is to ensure that the deliverables from suppliers meet the overall quality objectives of the organization. However for the purposes of the SLAs, keep the objectives simple and clear. The following are some recommended best practices when incorporating CISQ measures for application size and quality into SLAs.

1. Set Structural Quality Objectives

- **Quality objectives established in SLAs should depend on the business context.** The importance of different structural quality characteristics is usually determined by the potential consequences of problems. If a critical objective is to reduce IT costs, then Maintainability will be prioritized. If availability is critical, then Reliability will be prioritized. If confidentiality is critical, then Security will be prioritized. Where transaction volumes are high, Performance Efficiency will be prioritized. Hence quality objectives and targets should be determined by the business objectives and needs served by the application.

- **SLAs should include both baselines for quality levels that should be achieved or surpassed and minimum thresholds which are the lowest levels of quality that will be accepted without penalty.** All structural quality measures must be baselined, either through benchmarking or historical analysis, before the baselines and minimum thresholds are set. Based on the baseline scores and the requirements of the business, baselines and minimum thresholds are agreed upon between customers and their suppliers. Coding standards may also be defined to list architectural or coding weaknesses that should absolutely not appear in the source code.

- **Quality objectives should be stated simply and clearly.** For example, “Both Maintainability and Reliability scores in the next release of the application shall be improved by 10%. The quality violations

listed in Appendix A must not be introduced into the source code. At least 60% of the existing Severity 1 Security violations must be removed within the first 2 releases, and 95% removed within the first 5 releases." These expectations are based on benchmarks, are clearly stated, and can be tested at delivery.

- **Quality should be tracked and monitored for continuous improvement.** The indices should be collected and reported regularly to establish trends and identify issues before they become serious.

2. Measure Unacceptable Violations

In addition to specifying minimum thresholds and expected values of CISQ Quality Characteristic measures, SLAs should specify any unacceptable violations that must not exist in the code at delivery. An unacceptable violation (obligation) can either be from list of violations included in a CISQ Quality Characteristic or it can be a customer-defined violation. If any unacceptable violations are detected at delivery they must either be removed before a delivery will be accepted or submitted to a governance process, where financial penalties may be involved.

Unacceptable violations must be stated in sufficient detail that they can be detected through automated analysis of the source code. They should also be stated as coding standards so that suppliers can train their developers to avoid unacceptable violations at every step of the development process. An example of an unacceptable violation might be the detection of a transaction path from the user interface to the database that bypasses a secure user authorization function, thus violating a rule requiring proper authorization of all application users.

Unacceptable violations can be measured by a simple count of their occurrence in the source code at delivery. They can also be divided by the size of the application to establish a density metric. The number of unacceptable violations in each delivery should be tracked across deliveries to determine whether the supplier's performance is improving or if an improvement plan needs to be developed. The number of unacceptable violations should be tracked in addition to the other CISQ Quality Characteristic scores.

3. Use a Common Language

All quality measures to be used in SLAs should be explicitly defined, and have a description of how they will be measured. Here are some definitions that can be used in SLAs:

a. **Software Quality:** Software quality will be measured by the CISQ Quality Characteristic measures for Reliability, Security, Performance Efficiency, and Maintainability.
b. **Application Size:** Application size will be measured by the CISQ specification for Automated Function Points. The work performed on an application will be measured by the CISQ Automated Enhancement Point measure.
c. **Violations:** Application source code that violates a rule incorporated into one of the four CISQ Quality Characteristic measures or a coding standard incorporated into an SLA.
d. **Unacceptable Violations:** Violations specified in an SLA that must not exist in delivered code. All unacceptable violations must be remediated before delivery will be accepted.
e. **Minimum Threshold:** Minimum score established in an SLA that must be attained on one or more CISQ Quality Characteristic measures for a delivery to be accepted. Failure to achieve the minimum threshold can result in rejection of the delivery or financial penalties.
f. **Baseline:** The score established in an SLA that should be attained on one or more CISQ Quality Characteristic measures that the supplier should seek to achieve and sustain over releases. .

4. Evaluate Quality at the Application, Business Service or Transaction Level

Many analyses of software structural quality are performed at the component level. That is, the final quality measure is an aggregate of the quality scores for individual components. While this is an adequate indicator of *coding quality*, it does not assess the *architectural and structural quality* of the application as a whole. To assess the structural quality of the whole system, measures must be taken on quality rules that involve interactions among the different layers of an application, especially when they are developed in different languages and technology platforms. For instance, assessing the Reliability, Security, or Performance Efficiency of a transaction will require evaluating the interaction of components written in different languages and residing at different layers of the application such as the user interface, business logic, and database layers. As business applications include more and more distributed, Cloud and Mobile components, this becomes ever more important.

The most severe violations causing operational problems such as outages and security breaches typically reside in interactions among different application layers. Unless CISQ Quality Characteristics are assessed at the application level in addition to the component level, the customer will have a severely restricted view of application quality and will remain unaware of the possible presence of severe architectural violations. CISQ measures include rules for detecting architectural and other multi-layer, multi-language violations. SLAs should require that quality analyses be conducted on the entire as-built, multi-layer application rather than on just its individual components.

5. Define Roles and Responsibilities

In addition to defining SLAs, a contract should define the roles, responsibilities, and expectations of both the customer and the supplier. Even with a contract in place, customers and suppliers should never lose sight of the importance of sustaining a partnership in their relationship that aligns both parties in good faith to achieve the customer's needs and objectives while allowing the supplier to conduct an effective service.

Customer Responsibility:

- For each application under contract, the company should develop SLAs in consultation with the supplier that clearly define the minimum threshold (trigger levels) and baseline of measures, as well as all unacceptable violations that may not be in the source code at delivery.
- In their SLAs the company should clearly define the measures to be used in assessing these thresholds and targets and how the scores will be determined and/or calculated including the time frames for assessment.
- They should conduct a defined analysis and measurement process when code is delivered and provide a report to the supplier that clearly describes the results and any actions expected of the supplier. For discrete projects, payment may be withheld until these are achieved.
- Customers should host a post-analysis meeting with a supplier representative to discuss measurement results and plan for future actions and expectations regarding application size and structural quality.

Supplier Responsibility:

- The supplier should make all developers, testers, and other staff involved with the application aware of the SLA requirements and how they are to be achieved and sustained.
- For each application the supplier should commit to meet the minimum threshold and provide an action plan for achieving the baseline scores defined in the SLAs.
- The supplier should conduct quality assurance activities during the development period to ensure that

SLA agreements will be achieved.

- Upon request, the supplier should deliver whole source code including SQL structure definition files of the application to support an application audit by the customer, even if the new release does not modify all components of the application.
- The supplier should meet with the customer to discuss results of analysis and measurement activities, to identify any remedial action required immediately, and to plan for future actions and expectations regarding application size and structural quality.

6. Follow a Measurement Process

Establish a baseline: All of the CISQ measures of application size and structural quality should be baselined at the beginning of the contract period to provide a starting point for managing performance against SLAs.

Set baselines: Use the baseline application size and structural quality measures in conjunction with the requirements of the business to minimum thresholds and baselines. These thresholds and targets along with a list of unacceptable violations should be negotiated with the supplier and then incorporated into the supplier contract as SLAs.

Monitor and review results: CISQ quality measures should be generated according to a contractually defined process and schedule (e.g., once per month or release) by the customer and supplier. After the initial baselining of an application, the application size and Quality Characteristic measures are measured and compared with the minimum thresholds and baselines during the acceptance process for each delivery. Figure 1 displays an example chart tracking Reliability scores over several deliveries. Notice that the Reliability score for the initial delivery falls below minimum threshold, triggering remedial action. At release 7.5.1 the Reliability score achieves the baseline and it is expected that this score will be maintained or improved in all future releases.

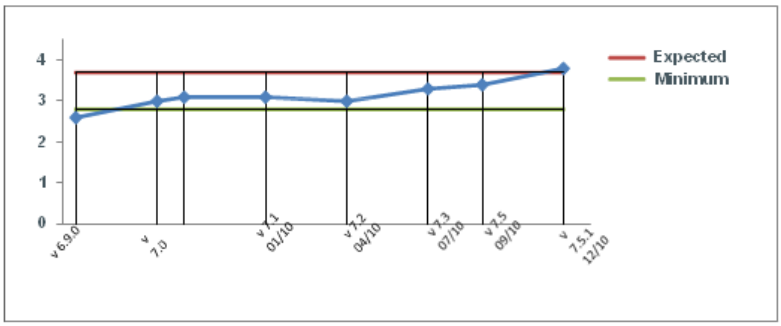

Figure 1: Example of tracking the Reliability of an application over releases

Determine remediation: For each release of the application, the supplier should commit to meet the minimum threshold for each Quality Characteristic that is the subject of an SLA. If the supplier fails to achieve a minimum threshold required in an SLA they should either conduct an immediate remediation or provide an action plan to achieve the threshold by the next release of the application. Customers should include contractual language that in addition to financial penalties, repeated failures to achieve minimum thresholds defined in the SLAs provides grounds for termination for cause. Conversely, customers may also wish to provide financial rewards for achieving expected scores ahead of schedule or improving beyond them.

Update SLAs: Three different events may justify changes to the SLAs.

- An addition to the Quality Characteristics, a change to the calculation process, or additions to the unacceptable violations list to be listed as obligations.
- The adoption of a new version of a technology or a new technology in the application that will directly impact the baselines or calculation of the Quality Characteristic scores.
- Experience with the application over several releases that indicates the baselines were not set at appropriate levels, such as when limitations or tradeoffs between Quality Characteristics are discovered during refactoring.

In the case of one of the three scenarios outlined, the source code of the application analyzed will be measured before and after the event. The difference between the two results will be presented to the supplier and if necessary the minimum threshold and baseline objectives in the SLAs should be updated through the governance process.

7. Use Rewards and Penalties Wisely

It is important that customers and suppliers review and establish the software quality objectives based on mutual agreement. A strong coercive approach may become counter-productive, and very often penalties are detrimental to the overall relationship between customers and suppliers. Good relationships require initial baselining, mutual agreement about reasonable goals in structural quality, enough time to improve a deteriorating system, and above all, fact-based evidence to assess progress. Consequently, an independent measurement system is a key factor in customer-supplier relationship success.

Since objective measurement systems provide facts for acceptance, rewards and penalties, and negotiations, they improve customer-supplier relationships by removing ambiguity, strengthening the quality of agreements, and removing subjectivity from the evaluation of results. As a result, measurable SLAs using CISQ measures for application size and structural quality should be incorporated into software-intensive customer-supplier contracts.