IT Quality: Measurement Implications for Large IT Projects in Texas

By Herb Krasner, Don Shafer and Linda Shafer, November 8, 2017

Executive Summary

The new Section 2054.159 of the Texas Government Code (HB 3275, 85R) adds specific monitoring requirements for Major Information Resources Projects (MIRPs) in Texas state agencies. It requires regular monitoring and reporting on project performance indicators of: schedule, cost, scope, and quality. Quality is believed to be the most challenging of the performance indicators, yet it has the largest potential for driving significant improvement. IT quality measurement is the focus of this report. There are both tactical and strategic reasons to measure quality on Texas' MIRPs.

Strategic - project and organizational process improvement

No tools, methods, or best practices will have any sustainable benefit if IT professionals are unable to perform their work in a disciplined environment. Achieving process maturity enables quantitative process management and other incremental improvement practices, which can then be applied to optimize or change the capability of a standard process, innovating when necessary. Certain quality metrics are proposed that facilitate this transformation, when improvement is a goal. Quality metrics are defined that support: process maturity, lean and agile development, adoption of standards, failure analysis, evaluation of newly adopted practices, and cybersecurity achievement.

Tactical - project and product tracking and control

The tactical application of IT quality metrics serves the following basic purposes:

- To help achieve project goals and objectives
- To help define and deliver the right product/system
- To assist in estimation, planning and executing the project effectively
- To help release a quality product/system at the right time
- To help analyze and evaluate created work products (intermediate and final)
- Evaluate the ROI using of good practices and standards

While these purposes may seem straightforward, in practice they are difficult to accomplish. Simply stated, IT quality metrics can help us learn to plan and manage MIRPs better. Metrics for the following typical work products are defined in this report: plan quality, requirements quality, architecture/design quality, software code quality, data quality, test quality, and operational system quality.

Adopting an effective metrics program and next steps

Along with measuring the other MIRP performance indicators (cost, schedule, scope) [1], a successful IT quality metrics program in an organization will mean that:

- The quality measurement program results are actively used in IT decision making
- IT quality measurements are communicated and accepted outside of the IT organization as well as within the project
- The quality measurement program will sustain itself and help overcome initial resistance to change
- Ultimately it will mean that a larger percentage of MIRPs will be considered fully successful

It will be important for agency CIO's and IT Leaders (e.g. IRMs, IT PMOs, IT project leaders, vendor contract managers) to begin the process of defining more precisely what their IT quality objectives and quality performance indicator(s) would mean; and laying out the framework for collecting and reporting on all newly required MIRP performance measurements. It will also be important for state IT leadership to update the various policies, procedures, rules, handbooks, reporting instructions, and templates that govern the proper monitoring of MIRPs in Texas, in preparation for the new law taking effect on January 1, 2018.

We are hopeful that the positions put forward in this paper provide useful guidance that can be adapted to the state, the agencies and local project needs as necessary. Please contact the authors listed in appendix 2 for more information or to answer questions about this and related documents.

IT Quality: Measurement Implications for Large IT Projects in Texas

By Herb Krasner, Don Shafer and Linda Shafer, November 8, 2017

Table of Contents

A. Introduction

Section 2054.159 of the Texas Government Code (HB 3275, 85R) adds new monitoring requirements for Major Information Resources Projects (MIRPs) in Texas state agencies. It requires regular monitoring and reporting on project performance indicators of: schedule, cost, scope, and quality. In our previous position paper, we discussed the implications and challenges of implementing this new law for Texas state and agency IT leadership[1].

In this position paper, it is our intention to specifically address the issue of measuring and monitoring **quality** on MIRPs, which we believe to be the most challenging of the performance indicators, yet it has the largest potential for driving significant improvement.

Quality is one of the four key IT project constraints (along with cost, schedule and scope), which need to be planned and controlled by effective project management over the entire IT project lifecycle. First of all, quality has to be defined in order to understand how to measure it. In the current state of affairs, IT quality is often left undefined, and therefore quickly becomes unmanageable, thus affecting the other key constraints.

In general, IT quality is the degree to which a system (or component) meets specified requirements and/or user/customer/stakeholder needs and expectations. But it is not necessary or desirable to wait until the delivered product appears to determine this. During development, we can say that IT quality has much to do with the acceptability of delivered work products over the life cycle of the project. With this general definition in hand, we can go about more precisely defining the acceptability levels of different kinds of delivered work products (e.g. plans, requirements, architecture, designs, code modules/increments, test plans, etc.). Usually this involves measuring the discovered *anomalies*/*deficiencies/defects* in designated work products.

Determining how well a system meets its functional and non-functional requirements as a measurement of quality often suffers from the assumption that you already have quality requirements – which is not usually the case. The typical way that problem is handled is to measure the rate of change and areas of uncertainty in the requirements and use that as a normalizing factor in relation to other quality characteristics.

In this paper, we highlight the two main purposes of measuring IT quality[2] on large projects:

1. Project and product tracking and control (tactical)
2. Project and organizational process improvement (strategic)

And we then identify and discuss reasonable quality measurements in each area.

---

[1] http://it-cisq.org/measuring-it-project-performances-in-texas-house-bill-hb-3275-implications/

[2] https://standards.ieee.org/findstds/standard/1044-2009.html - This standard provides a uniform approach to the classification of software anomalies, regardless of when they originate or when they are encountered within the project, product, or system lifecycle. Classification data can be used for a variety of purposes, including defect causal analysis, project management, and software process improvement (e.g., to reduce the likelihood of defect insertion and/or increase the likelihood of early defect detection).

B. Project and Work Products Quality

The following section discusses those typical work products that are produced and delivered during a large IT project. This section is not meant to imply a waterfall SDLC; therefore its concepts can be applied incrementally (as in agile) as well.

1. Plan quality

There are various plans associated with a large IT project that should be created and evolved. A **IT development plan** is a document, or several documents, that together specify goals, objectives, standards, practices, resources, specifications, roadmap and the set of activities relevant to a particular product, service, project, or contract.

There are industry standards to guide the content of an IT project plan (e.g. *Software Extension to the PMBOK® Guide*, 2013, and the PMI/Agile Alliance *Agile Practice Guide*, 2017). For example, a good IT development plan might contain the types of information shown in Appendix 1[3].

Evaluating plans is subjective, but they should be examined in terms of: acceptability, adequacy, feasible/achievable, well-refinedness, suitability, uncertainty/risk, likelihood range, appropriateness, value/ROI, etc. Having the whole team (including sponsoring executives) sign off on such a plan is a good sign of the commitment to achieving it.

Initially we want to know if a project has the *right* plan, resources, and supporting detail to execute successfully. Traditional approaches are to:

- Verify that the basic project management processes are in place
- Develop and train the project team to be successful.
- Establish performance/success objectives and related measurements

Basic project management processes and procedures are always critical to success. Some of the important plan areas to review and evaluate are:

- There is an approved Scope of Work (SOW) - essential for managing customer expectations and customer dependencies.
- The scope baseline is reasonably firm; with priorities, unknowns and assumptions identified.
- There is an approved financial baseline, including cost contingencies.
- There is a detailed project plan (schedule) that identifies all dependencies and milestones.
- The resource plan is complete. Specialized resources (e.g., architects and designers), contractors, and customer resources are identified and assigned to the project.
- A risk management plan is in place. There is a process to monitor and evaluate risks.
- There is a communications plan that includes the project team, the customer(s), and the project stakeholders.
- Appropriate performance reporting, issue management, and escalation processes are in place.
- The testing strategy is well defined. Testing experts are involved early in the project.
- Proper project controls exist for change control to identified artifacts.

---

[3] http://www.construx.com/Software_Development_Plan/

Ongoing evaluations during the project delivery phase include:

- Determining how well the project is tracking against the plan, contract, and customer expectations
- Determining if the deliverables are meeting the contractual obligations or defined commitments
- Regular schedule health checks[18], and especially when the plan is changing by more than 10%.

When the work goes off track relative to the plan, either the work needs to be adjusted or the plan needs to be adjusted. In either case, change control is important.

There is no doubt that measuring IT project plan quality is difficult, and the type and nature of the specific project will govern how solid the plan must be.

For many large IT projects a software quality plan is also required. Software quality planning includes:

- Defining specific quality goals,
- Determining which quality standards are to be used for both product and processes,
- Estimating the effort and schedule of software quality (e.g. SQA) activities.
- Identifying quality measures; statistical techniques; procedures for problem reporting and corrective action; resources such as tools, techniques, and methodologies; security for physical media; training; and SQA reporting and documentation.
- Defining quality verification and validation (V&V) activities.
- Identifying documents, standards, practices, and conventions governing the project and how these items are to be checked and monitored to ensure adequacy and compliance.

See SWEBOK@ V3.0, Chapter 10 for a more detailed description of the above.

Agile projects may handle software quality planning differently, for example, using RTF (running tested features) compared to acceptance criteria as its base quality metric. Retrospectives are also used to adjust the process to do better over the increments; and automated testing tools are heavily relied on.

In any case, it is important during planning to lay out defined quality objectives, standards and metrics. Quality doesn't just happen by itself.

2. Requirements quality

The purpose of having *requirements* is to form a common understanding and agreement between the stakeholders and the project team about delivering a system with specific functionality to their customers. This should include relative priorities and valuations. Historically this area has been the most problematic for larger IT projects[4].

This area varies greatly with the level of desired specificity of defined system objectives. Committed requirements/functions/features/use cases/user stories versus delivered results meeting defined "doneness" criteria are usually tracked. The term SMART is an acronym for Specific, Measurable,

---

[4] Krasner, H. (1989) Requirements Dynamics in Large Software Projects: A Perspective on New Directions in the Software Engineering Process. 11th World Computer Congress (IFIP89), August. Amsterdam, The Netherlands: Elsevier Science Publishers, B.V.

Achievable, Realistic and Time-bound. SMART is a much-used tool in determining if requirements are well written. Completeness and consistency are also important. Many measure *volatility* as a quality indicator.

In the conventional, waterfall model of development a standard like IEEE 830 could be applied.

That model requires a Software Requirements Specification to be correct, unambiguous, complete, consistent, ranked for importance, verifiable, modifiable, and traceable. Every one of these qualities could be (and should be) verified in their own specific ways. This view of requirements quality rarely applies to the innovative, agile IT projects typically done in IT shops today.

Content related requirements quality (as opposed to syntactical views) refers to goal model that connects expected content to project success factors. Such a model needs subject matter expert interpretation. That model can lead to a set of quality metrics to facilitate achievement.

Agile teams use a hierarchical model (aka roadmap) of themes, epics, and product backlogs of user stories to manage their requirements. Product owners prioritize the user stories and help create acceptance criteria. To enable delivering products with sufficient quality agile teams need to have user stories that are ready at the start of each sprint. Teams can use a Definition of Quality (DoQ) to check the user stories. A DoQ states the criteria that a user story should meet to be accepted into the start of an iteration. For example, using a model such as INVEST to grade the individual stories in six factor areas.

I – Independent
N – Negotiable
V – Valuable
E – Estimate-able
S – Small
T – Testable

Writing quality user stories is starting to emerge into a best practice, as we are now moving into the 2nd generation of disciplined agile methods.

Emerging research has started to define the contents of quality users' stories. This includes, for example, a set of criteria for user story quality. This includes *syntactic* (textual) criteria of: well formed, atomic, and minimal. This also includes the *semantic* (meaning) criteria of: conceptually sound, problem-oriented, unambiguous, and conflict free. This also includes the *pragmatic* (subjective interpretation) criteria of: full sentences, estimatible, unique, uniform, independent and complete. Tools are starting to emerge for analyzing user stories for these quality criteria.

Volatility/stability[5] – In agile, by committing to only what needs to be done in the next incremental delivery, the overall stability of requirements converges, usually resulting in a higher quality product delivered to customers.

[5] Lucas Sen, G., Dalpiaz, F., van der Werf, J.M.E.M. et al. Requirements Eng (2016) 21: 383. https://doi.org/10.1007/s00766-016-0250-x - https://link.springer.com/article/10.1007/s00766-016-0250-x

3. Architecture/design quality

IT system design has an all-pervasive impact on product quality. For instance, design decisions can (positively or negatively) impact various quality attributes.

An IT System Architecture[6] is the highest-level configuration of system components and the connections that coordinate component actions. Architecture is often the first artifact that represents decisions on how requirements of all types are to be achieved through functional partitioning and structural decomposition. Lower levels of design are typically driven by first principles such as: separation of concerns, reusable patterns, progressive elaboration, etc.

An IT architecture/design can be analyzed effectively using the three-view model):

- **Design principles:** Coarse-grained principles are abstraction, encapsulation, modularization, and hierarchy. The next level of granular principles maps to one of the coarse-grained principles; for example, granular principles such as generalization, localization, ordering, factoring, and substitutability map to the principle of hierarchy. Coupling, cohesion, and complexity map to modularization.
- **Constraints:** Project-specific constraints that the structure should adhere to such as language constraints, platform constraints, framework and library constraints, domain constraints, architectural constraints, hardware constraints, and process constraints.
- **Design quality attributes**: Quality attributes[7] such as: suitability, understandability, usability, changeability, extensibility, reusability, testability, security, efficiency and reliability. Hot spot analysis and violation detection fit here. Use of appropriate design patterns is also an important consideration.

Design quality is all about fitness for purpose, addressing the following questions:

- Does it do what is needed/required?
- Does it do it in the way that its users need it to?
- Does it do it reliably enough? Fast enough? Safely enough? Securely enough?
- Will it be beneficial?
- Will it be ready when its users need it?
- Can it be changed as the needs change?

Design quality is a measure of the relationship between the module and its application domain, which means that you might not be able to fully measure this fit until you place it into its intended environment. During design, we need to be able to predict how well the system will fit its purpose. Thus, we need to look for quality predictors. The two most commonly measured predictors of system architecture/design quality are simplicity and modularity.

Simplicity - the design meets its objectives and has no extra embellishments. This can be measured by looking for its converse, non-inherent complexity, such as seen in:

---

[6] Architecture: Form, Space, and Order, Francis D. K. Ching, Fourth Edition, Wiley, 2014.

[7] ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, ISO/IEC, 2011.

- Control flow complexity (number of paths through the program)
- Information flow complexity (number of data items shared)
- Name space complexity (number of different identifiers and operators)

Modularity - different concerns within the design have been separated into logical and relatively independent subunits. This can be measured by looking at:

- Cohesion (how well components of a module go together)
- Coupling (how much different modules have to communicate)

It should be noted that some agile approaches are not enamored of "upfront design" schemes. This is a limitation to be recognized when attempting to coordinate multiple agile teams working on different parts of the same system, and also when attempting to integrate multiple user story implementation units.

4. Software Code Quality

Source code is the most tangible artifact available for analysis. Once you have source code the following guidance applies.

The structure, classification and terminology of attributes and metrics applicable to software quality management have been derived from the ISO 25000 quality model (previously ISO 9126). Based on this model, the Consortium for IT Software Quality (CISQ) has defined four major desirable structural characteristics needed for a piece of software code to provide good business value. These characteristics are: Reliability, Efficiency, Security, and Maintainability. Software quality measurement quantifies to what extent a system (or code module) rates along each of these dimensions. The other four factors of quality from the ISO 25000 model do not yet have the level of definition required for standardized code measurement (i.e. functional suitability, operability, compatibility, and portability).

CISQ's quality model[8]

Even though "quality is a perceptual, conditional and somewhat subjective attribute and may be understood differently by different people", software structural quality characteristics have been clearly defined by the Consortium for IT Software Quality (CISQ). CISQ has defined the objectives that need to be achieved:

- Reliability

  An attribute of resiliency and structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software (its "stability" as termed by ISO). The goal for checking and monitoring Reliability is to reduce and prevent application downtime, application outages and errors that directly affect users, and enhance the image of IT and its impact on a company's business performance.

[8] http://it-cisq.org/standards/

- Efficiency

  The source code and software architecture attributes are the elements that ensure high performance once the application is in run-time mode. Efficiency is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount. An analysis of source code efficiency and scalability provides a clear picture of the latent business risks and the harm they can cause to customer satisfaction due to response-time degradation.

- Security

  A measure of the likelihood of potential security breaches due to poor coding practices and architecture. This quantifies the risk of encountering critical vulnerabilities that damage the business.

- Maintainability

  Maintainability includes the notion of adaptability, portability and transferability (from one development team to another). Measuring and monitoring maintainability is a must for mission-critical applications where change is driven by tight time-to-market schedules and where it is important for IT to remain responsive to business-driven changes. It is also essential to keep maintenance costs under control.

Software functional quality is defined as conformance to explicitly stated functional requirements, identified for example using Voice of the Customer analysis and/or documented through use cases, and the level of satisfaction experienced by end-users. The latter is referred as to as usability and is concerned with how intuitive and responsive the user interface is, how easily simple and complex operations can be performed, and how useful error messages are. Typically, software testing practices and tools ensure that a piece of software behaves in compliance with the original design, planned user experience and desired testability, i.e. a piece of software's disposition to support acceptance criteria.

The dual structural/functional dimension of software quality is consistent with the model proposed in Steve McConnell's Code Complete which divides software characteristics into two pieces: internal and external quality characteristics. External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not.

5. Data Quality

For most large IT systems, the quality of the system is only as good as the quality of the data that the system stores and uses. To make measures that represent "data quality" meaningful, we need a framework that relates the quality of user decision making to the quality of the data on which it is based[9]. The following guidance applies to persistent structured databases only.

[9] Arcady Maydanchik (2007), *Data Quality Assessment*. ISBN# 9780977140022

Data quality metrics are slightly different from code metrics due to their structural and behavior differences. Although they lack a standard theoretical framework, a pragmatic first principles approach is presented.

To achieve an organized approach for assessing data quality[10], data lineage levels should be considered:

- Terminology and business semantics (information requirements, business rule operands)
- "Type" - (metadata) (data model, e.g. relational schema)
- "Instance" - (e.g. strongly typed data elements)

At each lineage level data quality assessment results determine the: accuracy, completeness, consistency, precision, reliability, temporal reliability, uniqueness, validity, timeliness, relevance and cleanliness of the data. Each of those notional attributes can be defined more precisely as:

- Degree of agreement between a set of data values and a corresponding set of correct values
- Degree to which values are present in the attributes that require them
- Agreement or logical coherence in accordance with facts without variation or contradiction,
- The state of being exact, within the defined bounds and goals
- Logical coherence that permits rational correlation in comparison with other similar or like data
- Meanings and semantics that can change over time
- Data items that are provided at the time required or specified
- Data values that are constrained to a set of distinct entries—each value being the only one of its kind
- Conformance of data values that are edited for acceptability—reducing the probability of error
- Data values are current and not outdated or obsolete, measured from the perspective of the consumer of the data
- Percent of data appropriate to the execution of a given business process or processes, as defined by user input and output specifications.
- Cleanliness refers to the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty data.

Other models of data quality attributes are seen in ISO 8000-8:2015[11] and ISO/IEC 25012:2008[12].

The results of a Data Quality Assessment can provide operational metrics that can be used to continually monitor the quality of data. It is not unusual to find simple stoplight metrics or Likert scale metrics used for the different data quality attributes, which can then be aggregated in summary.

---

[10] Gerald Weinberg (1993), *Software Quality Management, Volume 2 – First Order Measurement*. ISBN# 0932633242

[11] *Benson, Peter (2009), "ISO 8000 Data Quality — The Fundamentals, Part 1", Real-World Decision Support (RWDS) Journal, **3** (4)*

[12] ISO/IEC 25012:2008 "Software engineering -- Software Product Quality Requirements and Evaluation (SQuaRE) -- Data quality model"

6. Test quality

Testing is quite often the costliest SDLC activity, and consists of both static validation and dynamic verification that a program provides the expected behaviors. In many large projects, poor project execution causes testing to become squeezed as the delivery date approaches. This creates more errors detected after release, which are far more expensive to correct and also result in dissatisfaction among system users.

Various types of tests are used to ensure the quality of the emerging product. Those tests typically include: unit tests, integration tests, functional tests, systems tests, regression tests, stress tests, performance tests, UI testing, security penetration testing, alpha/beta tests and final acceptance tests. The active testing process is usually preceded by requirements analysis, test planning, testing infrastructure establishment, and test database development. Defining the quality objectives for testing helps to focus the overall testing effort, and helps to determine when testing is really *done*.

The one clear measurement of the quality of testing is the number and severity of the bugs discovered and fixed prior to release or passed onto the next phase of the SDLC. Potential test quality metrics derived from the data collected during testing and operation are:

- Defect leakage (opposite of removal efficiency) into fielded system or next phase
- Cost of Quality (CoQ) *See [Krasner, 1998]* for example definition[13]
- Test effectiveness as a percentage value of the difference between the number of defects found by the test process, and the overall defects found for the system.
- Testing progress indicators (e.g. failure/passage rates, test coverage)
- Defect counts over time, defect density with respect to system size
- Serious incident rate
- Delivered quality functions versus "doneness" criteria
- System reliability indicators: e.g. MTTF, MTBF, MTTR, availability

There is a wealth of quality data created during the testing process to serve as the basis for quality performance indicators.

7. Operational system quality

User satisfaction, cost of ownership, cost of quality and technical debt are all useful measures of deployed IT system quality. They are presented below in order from broadest to narrowest concept. All are fairly well defined in the literature but rarely measured in practice. In summary, they are:

- User/customer satisfaction
    - Addresses how well we have achieved the desired level of satisfaction and user perceived quality
- Total Cost of Ownership (TCO)

[13] Krasner, H. (1998) *Using the Cost of Quality for Software*, in CrossTalk, The War on Bugs, Vol. 11, No. 11, November 1998, pp 6-11, see online at www.stsc.hill.af.mil/Crosstalk/crostalk.html

- A financial estimate intended to help owners determine the direct and indirect costs of a system over its intended life cycle
- When incorporated in any financial cost/benefit analysis, TCO provides a basis for determining the total economic value of an investment (e.g. ROI)
- TCO tries to quantify the financial impact of developing, deploying and maintaining/supporting an IT system over its entire life time

- Cost of Quality (CoQ)
  - The cost of NOT creating a quality product or service.
  - Represents the difference between the actual cost of a system or service and what the reduced cost would be if there were no possibility of substandard work, system failures or defects in their development
  - Includes defect prevention, find & fixes (pre- and post-delivery), technical debt, rework, damages and warranty costs
- Technical Debt
  - A metaphor referring to the eventual consequences of poor system analysis, architecture, design and/or development within a given codebase
  - The debt can be thought of as work that needs to be done before a particular job can be considered complete. If the debt is not repaid or serviced, then it will keep on accumulating interest, making it harder to implement changes later
  - Unaddressed technical debt increases software entropy (disorder)
  - One possible Technical Debt metric utilizes CISQ's Automated Quality Characteristic Measures to measure the impact of critical violations of good coding and architectural practice in the source code of software. The Technical Debt specification was formally approved as a standard by OMG. See http://www.omg.org/spec/ATDM/

The recorded and projected cost of operational deficiencies/defects provides one simplifying measurement concept that allows one to estimate all of the above. However, that approach typically comes too late to be an effective early warning technique.

C. Process Improvement Considerations

It is our intention, in addition to enabling quality performance on IT projects, to also encourage organizations to continuously improve their performance using these same metrics to baseline and then track their own progress.

1. Process Maturity Framework[14]

It is asserted that, no tools, methods, or best practices will have any sustainable benefit if professionals are unable to perform their work in a disciplined environment. Process maturity implies that the first step in improvement is to stabilize the local working environment so that professionals can perform disciplined work. Once they have a chance to see what practices work best they can then standardize them and achieve an economy of scale across the larger organization. At this point, quantitative process management and other incremental improvement practices can be applied to optimize or change the capability of the standard process, innovating when necessary. Quality metrics facilitate that transformation, when improvement is a goal[15].

2. Lean and agile movements in the IT industry

Lean IT is now fashionable. The practices comprising this system continually improve business results by focusing on reducing process inefficiencies and defects to drive waste out of the value chain. Processes that fail to contribute either directly or indirectly to the value should be eliminated or re-conceived. Quality metrics can contribute to this approach.

Agile IT describes a set of values and principles for adaptive development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. It advocates incremental planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. These principles support the definition and continuing evolution of many modern software development methods. Specific techniques (e.g. refactoring) are often used to improve quality and enhance product development agility. Quality metrics can contribute to any of the specific methods found under this scheme.

Other related quality movements are also found in various places (e.g. TQM, Baldrige, 6-sigma, etc.) Obviously quality metrics support those initiatives as well. We often see Lean IT being pursued at the organizational level, while agility is often pursued at the project level. They work well together.

3. Industry Standards

The nice thing about standards is that there are so many to choose from. Many of the current IT/software standards appear to be focused at the "good practices" level from particular points of view, boiled down from a broad range of industry experiences. In this document, we are not advocating any particular standard, except ISO 25000[16] to help focus on the subject of IT quality definition and measurement.

It is crucial that every large IT project create a working definition of quality during the planning process, which serves as a basis for measurement over the project's lifecycle. Standards that help to formulate such a definition are the most useful.

---

[14] Curtis, B. (2011) Disputation of Misinterpreted Principles Underlying the Process Maturity Framework, August 1, 2011, European SEPG

[15] http://cmmiinstitute.com/capability-maturity-model-integration

[16] https://www.iso.org/standard/64764.html

4. Cybersecurity Measurement

Cybersecurity is a pressing problem area for today's IT systems. It is typically viewed as a subcategory of quality. The goals of IT Cybersecurity Performance Measurement are to:

- Strengthen the foundations of delivered systems,
- Help identify, prevent, detect, handle cybersecurity incidents,
- Identify and replace unreliable development and maintenance practices with proven best-of-breed practices;
- Identify and enable operational metrics accounting for the evolution and increased complexity of COTS software and hardware;
- To enable continuous improvement of the operational system.
- Justify the costs/benefits of defensive mechanisms (e.g. extra code). When measured carefully, these costs may prove to be negligible, as they have been in most cases covered in experiments done at NIST thus far.

The current approach to cybersecurity measurement is to use enhanced risk management as the basic approach. This seems reasonable as we continue to develop potential new metrics in this area. Measuring vulnerabilities and resilience[17] is the next logical step.

Cyber-resilience is a family of related ideas, not a single thing. It's not just a technical system property. We define it as the ability to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks, or compromises on cyber resources (MITRE, 2016). This includes achieving:

- System properties related to being able to withstand attacks and deviations from the intended state and go back to a pre-existing, or a desirable or acceptable, state.
- Ability to deliver, maintain, improve IT services when facing cyber threats and related evolutionary changes

Possible measures that characterize cyber resilience are:

- *Dependability/availability* in the presence of flaws, faults, defects, vulnerabilities
- *Amount of disruption* that a system can tolerate under attack
- *Probability of correct service* given that an attack occurred
- *Set of probabilities* of the "levels of accomplishment" of a system's function under attack

Current data sources for those measures include**:**

- Identified:
  - Risks; the default model
  - Threats (surface, vectors, scenarios, etc.)
  - Vulnerabilities (CVSS 3.0, Mitre CWE, CISQ Security metric, CERT)
- Cyber security technical debt (a definition has yet to be determined)
- Data from: antivirus and antispyware tools, intrusion detection systems, firewalls, patch management systems, security logs, and vulnerability analyzers.

---

[17] K. Wolter et al. (eds.), Resilience Assessment and Evaluation of Computing Systems, Ch. 1, L. Strigini, DOI: 10.1007/978-3-642-29032-9_1, © Springer-Verlag Berlin Heidelberg 2012

Cybersecurity is a timely and important topic, and deserves to have an expanded dialogue of its own. A future position paper will cover this in the detail it deserves.

D. Next Steps

Managing a MIRP for success is challenging. Effective measurements can help, especially in the area of IT quality. We assert that IT quality must be defined and measured if significant improvement is to be achieved. We have observed that it is often the case that IT quality on a MIRP is ambiguously defined, or not at all. Quality is a multi-dimensional concept reaching into: the system of interest, stakeholder viewpoints on that system, and the quality attributes of that system. There are also different levels of abstraction to consider, such as quality in the broad sense (common language) versus quality in the very specific sense (IT professional view). Unifying the various views through IT quality definition and measurement would be a good start.

It will be important for state agency CIO's and IT Leaders (e.g. IRMs, IT PMOs, IT project leaders, vendor contract leaders) to begin the process of defining more precisely what their IT quality objectives and quality performance indicator(s) would mean; and laying out the framework for collecting and reporting on all required MIRP performance measurements. In preparation for the effective date of the new law (Jan. 1, 2018), it will be important for state IT leadership to update the various policies, procedures, rules, handbooks, reporting instructions, and templates that govern the proper monitoring of MIRPs in Texas.

In many leading edge organizations, it is hard to imagine managing IT development and evolution without metrics to help guide decision-making. There are many practical uses of IT measurement, but four areas in particular deserve our attention:

1. Project estimation, progress monitoring and corrective action
2. Evaluation of work products
3. Quality-driven process improvements
4. Experimental validation of best practices

We are hopeful that the positions put forward in this paper provide useful guidelines that can be adapted to the state, the agencies and local project needs as necessary.

"Quality is not free, but it is way cheaper than the alternative"

To the extent that other states are following a similar path to effective measurement of large IT projects, perhaps the guidance contained herein may help them as well.

---

18. Krasner, H. et al, Special Report on A Schedule Health and Risk Analysis Of the OAG T2 Integrated Work Plan, Version: 1.0 Final, 5/13/2016

Appendix 1: Software Development Plan Template

Revision Chart

Preface

Contents

1. Introduction
    - 1.1 Project Overview
    - 1.2 Project Deliverables
    - 1.3 Evolution of the Software Project Management Plan
    - 1.4 Reference Materials
    - 1.5 Definitions and Acronyms
2. Project Organization
    - 2.1 Process Model
    - 2.2 Organizational Structure
    - 2.3 Organizational Boundaries and Interfaces
    - 2.4 Project Responsibilities
3. Managerial Process
    - 3.1 Management Objectives and Priorities
    - 3.2 Assumptions, Dependencies, and Constraints
    - 3.3 Risk Management
    - 3.4 Monitoring and Controlling Mechanisms
    - 3.5 Staffing Plan
4. Technical Process
    - 4.1 Methods, Tools, and Techniques
    - 4.2 Software Documentation
    - 4.3 Project Support Functions
5. Work Packages, Schedule, and Budget
    - 5.1 Work Packages
    - 5.2 Dependencies
    - 5.3 Resource Requirements
    - 5.4 Budget and Resource Allocation
    - 5.5 Schedule
6. Additional Components
7. Index
8. Appendices

Appendix 2: Authors' Contact Information

Primary author: Herb Krasner
Email: hkrasner@utexas.edu
Phone #: 830-693-0631
Bio: http://www.ece.utexas.edu/people/faculty/herb-krasner

Author: Don Shafer
Email: dshafer@athensgroup.com
Phone #: +1 512.663.1459
Bio: https://www.linkedin.com/in/donshafer/

Author: Linda Shafer
Email: lshafer@computer.org
Phone #: +1 512.474.4398
Bio: https://www.linkedin.com/in/linda-shafer-15a0218/