Software Engineering Institute | Carnegie Mellon University

Measures and Measurement for Secure Software Development

Carol Dekkers
Dave Zubrow
James McCurley

February 2007

ABSTRACT: This article discusses how measurement can be applied to software development processes and work products to monitor and improve the security characteristics of the software being developed. It is aimed at practitioners—designers, architects, requirements specialists, coders, testers, and managers—who desire guidance as to the best way to approach measurement for secure development. It does not address security measurements of system or network operations.

OVERVIEW

This practice area description discusses how measurement can be applied to software development processes and work products to monitor and improve the security characteristics of the software being developed. Measurement is highly dependent on aspects of the software development life cycle (SDLC), including policies, processes, and procedures that reflect (or not) security concerns. This topic area is aimed at practitioners—designers, architects, requirements specialists, coders, testers, and managers—who desire guidance as to the best way to approach measurement to monitor and improve the security characteristics of the software being developed. It does not address security measurements of system or network operations, nor does it address an organization's physical security needs.

MEASUREMENT AND THE SOFTWARE DEVELOPMENT LIFE CYCLE

Measurement of both the product and development processes has long been recognized as a critical activity for successful software development. Good measurement practices and data enable realistic project planning, timely monitoring of project progress and status, identification of project risks, and effective process improvement. Appropriate measures and indicators of software artifacts such as requirements, designs, and source code can be analyzed to diagnose problems and identify solutions during project execution and reduce defects, rework (effort, resources, etc.), and cycle time. These practices enable organiza-

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

tions to achieve higher quality products and reflect more mature processes, as delineated by the CMMI.[1] Watchfire has published a short description of typical application security activities for each level of the CMMI [SLDC areas related to the definition and use of measures for secure development addressed in the Build Security In modules include

1. Requirements Engineering
2. Architectural Risk Analysis
3. Assembly, Integration, and Evolution
4. Code Analysis
5. Risk-Based and Functional Security Testing
6. Software Development Life-Cycle (SDLC) Process
7. Coding Rules
8. Training & Awareness
9. Project Management

Risk management in general is addressed separately in the module Risk Management Framework. In contrast to the traditional focus of risk management on project failure in software development, it must now be extended to address the malicious exploitation of product flaws after release and throughout maintenance. Threat modeling and its use in the SDLC is addressed in the Attack Patterns content area. All of these areas are positively impacted by the use of measurement.

SOFTWARE ENGINEERING MEASUREMENT PROCESS

Recent work to establish a common perspective on how to perform software measurement and analysis can be found in International Organization for Standardization and International Electrotechnical Commission (ISO/IEC) 15939 (Software Measurement Process standard), the Capability Maturity Mode®l[2] IIntegration (CMMI) Measurement and Analysis process area, and the guidance provided by the Practical Software and Systems Measurement (PSM) project. For purposes of description, the practices from the CMMI® model are presented

---

[1] CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[2] Capability Maturity Model is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

here. The practices are organized around two major goals or themes: aligning measurement and analysis activities with organizational and project goals and then performing the measurement and analysis activities. Briefly, the practices for aligning measurement are

- Establish Measurement Objectives (Goals)
- Specify Measures
- Specify Data Collection and Storage Procedures
- Specify Analysis Procedures

The practices for performing measurement are

- Collect Measurement Data
- Analyze Measurement Data
- Store Results and Data
- Communicate Results

These practices, shown as steps in Table 1, are important for several reasons: (1) goal driven measurement is an important first step to ensuring management commitment to the measurement initiative, (2) the organization and/or project is forced to target and measure the items necessary to meet the objective(s), and (3) measurement enables greater success by providing a framework where decisions and process improvement can occur through the analysis of data. The following steps call for the organization and project to plan their measurement activities so that the right measures are collected, analyzed, and communicated to the appropriate people in an informative format and timely manner. Project management and insight into specific aspects of product quality depend on data that is relevant, reliable, current, and valid. Following these practices (or steps) focuses the measurement activities on the collection of data that will be used, rather than simply collecting data for the sake of measurement.

With respect to the development of secure software, it is important that security concerns be clearly identified and addressed in all steps of the measurement and analysis process outlined below.

Table 1. Measurement and Analysis Process

Step Number	Step Name	Input	Techniques	Critical Participants	Output

1	Establish Measurement Objectives (Goals)	Software/system requirements	See Requirements Engineering; also elicitation practice area	Stakeholders, requirements team	Agreed-to measurement objectives (for which the attainment can be measured)
2	Specify Measures	Measurement objectives, SDLC	Facilitated work sessions	Measurement analysts, process engineers, security subject matter experts, users/customers	Measurement definitions for security; focus on problem-prone modules; known vulnerabilities; define needed security levels
3	Specify Data Collection and Storage Procedures	Measurement definitions, SDLC	Procedure /process mapping (and potentially design if a current void)	Process engineers, designers, practitioners	Process changes, training needs, tool needs
4	Specify Analysis Procedures	Measurement objectives and definitions (GQM)	Literature review, elicitation	Process engineers, measurement analysts, security experts	Identified statistical and/or qualitative analytical techniques
5	Collect Measurement Data	Measurement plan, data collection tools and infrastructure, instrumented processes	Automated tools and manual forms associated with artifact inspections and testing	Practitioners, testers, measurement analysts, quality assurance	Data in usable form (e.g., database, spreadsheet)
6	Analyze Measurement Data	Output of Step 5	Specified in Step 4	Measurement analysts	Summary, graphical displays, detailed results

7	Store Data and Results	Outputs of Steps 5 & 6	Inspection database or test results database	Measurement analysts, database administrators	Retrievable source data and analytical results
8	Communicate Results	Analyst summary, graphical report	Formatted results with interpretation and recommendations	Project engineers, project/line management, security experts	Feedback to development team, program manager

Effective use of the above process relies first on agreeing on the desired security characteristics and the importance of achieving the resultant measurement objectives, which can be applied to both product and the development process. These goals rely on explicit system requirements—which means that security aspects must be specified early. The organization should assess the risk environment to address probable threats and translate these concerns into specific requirements addressing security as well as design and implement a development process that will ensure the "building in" of such requirements.

After security-related requirements of the product are specified, measurement objectives may be formulated that will provide insight into achieving the security requirements. Examples of analytical questions which lead to measurement objectives include the following:

- What vulnerabilities have been detected in our products? Are our current development practices adequate to prevent the recurrence of the vulnerabilities?
- What process points are most vulnerable to the introduction of security-related risks (e.g., injecting reused code/modules into programs—where the variables could go unchecked, etc.)?
- What proportion of defects relate to security concerns and requirements? Do defect classification schemes include security categories?
- To what extent do practitioners comply with security-related processes and procedures?
- To what extent are security concerns addressed in the intermediate work-products (requirements, design, etc.)? Have measures associated with security requirements and their implementation been defined and planned?
- What are the critical and vulnerable modules? Have vulnerabilities been identified and addressed?

Threat modeling, or the attempt to identify likely types and sources of attacks, can also form a significant guiding requirement to the development processes for secure products. A recent thesis by Stuart E. Schechter at Harvard's Department of Computer Science uses economic models for valuing the discovery of vulnerabilities in the final or end product during development [Schechter 2004]. His measurement of security strength depends most on threat scenarios to assign values to vulnerabilities in an effort to extend a market approach to the development process. Many risk and threat methodologies are available publicly, and Microsoft has published extensive materials that delineate the company's approach to analyzing and mitigating threat risks during the SDLC [Microsoft 2003, MSDN 2004].

PROCESS MEASURES FOR SECURE DEVELOPMENT

Process artifacts that implement security measurement objectives for the development process should address

- the existence of security policies applicable to the SDLC (roles, procedures, responsibilities, management, coding rules, acceptance/release criteria, etc.)
- compliance to the above
- efficiency and effectiveness over time

The security measurement objectives for the development process are identical to general measurement objectives—they need to be included in the process implementation. Such measures could be implemented as part of an organization's integrated quality assurance function.

Although targeted for systems development and risk assessment as a whole, useful guidance for measurement of this type can be found in the NIST publication Security Metrics Guide for Information Technology Systems [Swanson 03]. Risk management can encompass secure coding and provides a familiar framework to incorporate new practices and procedures to address software security issues. As mentioned in Software Security: Building Security In [McGraw 2006], tracking risk throughout the life cycle of a software development project affords managers and analysts the ability to assess relative measures of risk improvement.

The least expensive approach to software development dictates that flaws/defects are identified as early as possible in the life cycle. Requirements analysis typically addresses the functional aspects of the product, but with security in mind, additional analysis of non-functional requirements must also be used to identify security concerns. Security requirements often take the form of what is not sup-

posed to occur but can still be tracked to closure in the same manner as other requirements. Security requirements can also be a mix of both functional and non-functional requirements. Threat modeling is especially important to this phase of the life cycle since it can help in the preparation of test strategies and use cases. This risk-based view of development carries through the design phase with new insight into architectural concerns (see Architecture module). Proper design incorporating security that is implemented correctly as the code is constructed minimizes the attackability of the final product. Again, effective code policies can be tracked for compliance during design and through the remainder of the life cycle.

Testing schedules have often suffered in past projects as the need to deploy the product overrides other concerns. Security ramifications are changing this view due to the potential impacts of the attackability of the final product. The functional testing practices of the past prove insufficient to deal with non-functional security issues. Addressing security up front means having a test strategy throughout the life cycle where security issues are addressed in each phase and are not passed on to the next phase of product development. As mentioned above, this requires tracking bugs early on, but it also requires security test planning at an early stage and confronts risk issues identified in the requirements and design stages.

Measurements prove valuable when they are useful, key components of the development effort, as opposed to mere status reports. To achieve usefulness throughout the life cycle, however, everyone involved must understand the measurement's definitions and uses. Appendix A includes a measurement indicator template for documenting the key attributes of each indicator (e.g., measures used to construct the indicator, algorithm used, assumptions, etc.). The template has found wide acceptance for documenting the indicators used to implement software engineering measurement and can be used for new security measurement purposes. It forms a fundamental building block for any measurement program and, over time, allows the organization to catalog its metrics definitions and enables trend analysis. As an organization gains experience in building secure software, such trend analyses provide useful feedback to project managers about the efficacy of each process. More than that, trends also identify the effectiveness of policies, tools, and techniques and also allows for better estimation of all engineered parameters, including security.

Defect density is a commonly used measure of product quality. It is often computed as the number of defects discovered during system test or during the first six months of operational use divided by the size of the system. Estimates of defects remaining in the product (calculated by techniques such as phase containment, defect depletion, or capture-recapture techniques) form a natural analogue

to estimate remaining security vulnerabilities in the software. Phase containment of defects refers to an analytical technique that measures the proportion of defects originating in a phase that are detected within that same phase. It provides a good characterization of the ability of the development process to maintain quality throughout the SDLC. The INFOSEC Assurance Capability Maturity Model (IA-CMM) recognizes the impact of quality control by listing "Establishing Measurable Quality Goals" as one of two features that enable a level 4 rating of Quantitatively Controlled [NSA 2004].

PRODUCT MEASURES FOR SECURE DEVELOPMENT

In the product context, security concerns addressed by measurement objectives may take the form of

security requirements, which are based on risks determined by threat assessments, privacy policies, legal implications, and so forth and can be specified as to extent and completeness

1. architecture security, which addresses the specified security requirements
2. secure design criteria, where security requirements can be traced
3. secure coding practices, where integrity can be assessed and measured

Not all measures need to be complicated. Measures should be as simple as possible while still meeting information needs. For example, in the requirements phase it is useful to know whether security-related concerns have been included in defining system requirements. This could be measured initially as yes or no. As experience with the measure accrues over time, the measure could evolve to characterize the extent that requirements have been checked and tested against security concerns. Determining the extent that security measurement objectives are implemented during the design and coding phases will make use of tools as well as inspections or reviews. Many of the inspection measurements will be in the form of traditional defect identification checklists, to which security-oriented items have been added. Table 2 lists some sources of vulnerabilities or concerns that have been widely documented, along with a reference to the part of ISO/IEC 9126 that has defined a relevant measure. Software inspection checklists could be extended to include review of the issues in the table.

For instance, one could track the percentage of sources of input that have validation checks and associated error handling. That is, checking each input source for length, format, type, and so forth and its associated exit flows—either accepted then executed or as an error/exception and not executed. The target for this

measure would be 100%, unless performance suffers unacceptably as a result, or it would cost too much to implement. Note that while this simple measure is an improvement over no measurement for this type of vulnerability, it does not address the potentially complex issue of determining the effectiveness of an input validation technique as implemented and whether any particular datum should be counted in the tally. This would require ongoing tracking of this measure's performance to characterize the effectiveness of the input validation techniques used. Over time, the organization can benchmark these kinds of measures as performance standards.

Table 2. General Code Integrity Issues

- access control
  - access controllability (ISO 9126-3)
  - access auditability (ISO 9126-3)
- input validation – particularly to address buffer overflows, format string attacks, SQL injection, etc.
- exception handling/error traps (log bad entries, no execute)
- resource management - consumption, retention, race conditions, closure, etc.
- privileges management – principle of least privilege
- system calls, process forks, etc.
- unexpected behavior or system response
- data security issues
  - data security levels (proprietary, classified, personal, etc.)
  - data encryption (ISO 9126-3)
  - data corruption prevention (ISO 9126-3)
- garbage handling/memory management
- risk analysis (identified risks, ranked, with impact analysis, and mitigation and fallback plans)
- implementation bugs
- architectural flaws

Web Applications

- scripting issues
- sources of input
- forms, text boxes, dialog windows, etc.
- regular expression checks
- header integrity
- session handling
- cookies
- framework vulnerabilities (Java, .NET, etc.)
- access control: front and back door vulnerability assessment
- penetration attempts versus failures
- depth of successful penetrations before detection

Simple measures of enumeration and appropriate security handling for vulnerabilities would provide insight into the security status of the system during development. In addition to the above table, a useful list of "Measurable Security Entities" and "Measurable Concepts" has been published by Practical Software and Systems Measurement [PSM 2005].

The following questions generated by the PSM/DHS Measurement Technical Working Group address many of the above issues and can provide starting points for developing measurement objectives:

Planning:

- How much of the system incorporates security features?
- What is the current state of completion of planned security tasks, and how is it measured?
- Do I have qualified people, tools, and environments?
- Do I have qualified tools that support security analysis during development?
- How effective are the security tools?
- How effectively are the tools used?
- How many (insecure) code practices do these tools identify?
- How much does it cost to implement security aspects?
- What is the scope of the security-critical functions, system, and software?
- What are the potential losses/damages?
- What are the external threat agents?
- What security risks are covered by financial means?
- How badly can I be harmed if the system is violated? If something happens, how much is lost or harmed? How valuable is the data? What is the level of criticality? How is it measured?

Requirements phase:

- Are we following the best practices when expressing (security) requirements?
- Are the security requirements valid? Do they meet user needs?
- Have we traced our security requirements?
- Have all sources been considered (e.g., threats, assets, usability, certification)?
- Have all stakeholders been considered?
- Can I tell if it a security requirement has been satisfied?
- Are security requirements completed on schedule?
- Have requirements been deliberately changed?
- What is the rate of change of security requirements?

- Does each security requirement trace to an appropriate design unit(s)?

Design:

- Have we followed security design principles?
- Is the design sufficiently detailed to meet the security requirements placed on it?
- Can the design be analyzed to verify that it meets the security policy and requirements?
- What external systems and interfaces does this system depend on for security risk mitigation?

Coding:

- Has each unit complied with the secure coding practices?
- Have bugs been identified, classified, and traced to requirements?
- Do we have adequate coverage of security in user aids (help files, manuals, training, etc.)?

Testing:

- Have we completed security testing (e.g., attacks, penetration)?
- Have all identified security issues been resolved?
- How broad is the security testing?
  - Does it include tools and people?
  - How many attack patterns are evaluated?
  - Are we testing at the unit, subsystem, system-of-system level?
  - Is the testing static or dynamic?
- What is the scope of the attack surface?
- Have we provided sufficient information with defects to allow prioritization, root cause analysis, and remediation of vulnerabilities?
- In resolving non-security defects, have we introduced any security issues?
- What is the current progress of evidence development?

The following measures from a variety of sources have been suggested as useful:

- number of security defects discovered in-house versus in the field
- number of security defects detected in strategy or design versus in the field (repeat for each phase)
- predicted versus actual labor costs for fixing defects at each stage of development
- security defects per thousands of lines of code (KLOC)

- number (and percent) of security defects considered low/medium/high/critical
- number (and percent) of security defects fixed
- average or median time (and cost) to fix each defect
- quartile rankings for each developer group, based on defects/KLOC and average or median time to fix, ranked by severity

And finally, questions the organization should address as a whole:

- Are risk mitigations on schedule?
- How many known vulnerabilities exist in the system? How many have been resolved or accepted as risks documented and transmitted to the customer?
- Have we followed the regulatory requirements?
- Have we followed the organization's standard process model?
- Have we followed the relevant best practices?
- Have we followed the relevant policies?
- Have we modified our process to detect new potential threats?
- How much residual security risk exists in the operational system? How much confidence do you have in this answer?
- Have we created any unintended security consequences with anything else we have done?
- How efficiently has the security investment been used?
- Have my choices in security improvements been well timed, well spent, and appropriate?
- Are my customers satisfied with the product's security?

These questions can form a solid basis for measurement in most development organizations, regardless of size or methods employed. They are presented here to further the discussion of what constitutes adequate measurement to address security issues during development, and should not be interpreted as exhaustive. Each question requires some extensive, non-trivial work to come up with agreed upon definitions before it can be measured.

COMMUNITY OF INTEREST

The continuing onslaught of software systems by malicious actors has prompted a great deal of activity. Every week there are new stories of compromised systems, yielding private information. Recently, we've again seen cyberwar activities that raise new concern for national security. And we still see the deployment of new software-intensive systems that do not perform as intended and that enable exploitation.

In the last few years, the development community and the acquirers of software systems have initiated several collaborative efforts aimed at improving the trustworthiness of software. One such effort has resulted in the development of a Software Assurance Evidence Metamodel (SAEM). The published specification defines terms and characterizes software assurance evidence that can be used for judging whether a particular software system fulfills a given set of requirements. Although this is only the first in a series of specifications, it represents a promise leading to the creation of new tools related to software assurance. Evidence is defined as facts, which are grouped in the following categories:

- software artifacts
- software operational environment
- methodologies
- development process
- people
- development environment
- regulatory compliance controls

This effort is being led by Adelard LLP, KDM Analytics/Hatha Systems, Lockheed Martin, Computer Sciences Corporation, and Benchmark Consulting and is supported by the University of York, MITRE, and the SEI.

Another broad-reaching effort was initiated by the U.S. Department of Defense (DoD), Department of Homeland Security (DHS), and National Institute of Standards and Technology Software Assurance (SwA) Measurement Working Group. The Practical Measurement Framework for Software Assurance and Information Security resulting from the effort of industry, government, and academic collaborators will be published online in October 2008. Given the long term nature of the collaboration, the Practical Measurement Framework leverages several useful resources that have become available, particularly the Common Vulnerabilities and Exposures (CVE), Common Control Enumeration (CCE), Common Weakness Enumeration (CWE), and Common Attack Pattern Enumeration and Classification (CAPEC).

The Common Vulnerabilities and Exposures (CVE) list is probably the best known of the above resources, in that it has gained widespread agreement and adoption. It is a valuable resource that provides the community with an ability to communicate effectively about vulnerabilities of software systems. Begun in 1999, the dictionary currently lists about 6,000 publicly known vulnerabilities. The Common Weakness Enumeration (CWE) is of particular interest to the development community, since it lists some 605 (as of Sept 2008) weaknesses in source code and operational systems related to architecture and design. This in-

sight can lead to useful measurements regarding assurance quality and compliance and also to the development of new tools for building and evaluating software.

The Common Attack Pattern Enumeration and Classification (CAPEC) addresses the need to identify how vulnerabilities are exploited by giving the community a "firm grasp of the attacker's perspective and the approaches used to exploit software." Knowledge of attack patterns can be especially useful during development activities to build defense into the software system. These resources and more have been established by MITRE and can be accessed directly or through their website, Making Security Measurable. The Practical Measurement Framework provides measurement insight utilizing these and other resources such as ISO/IEC standards 15939, 16085, 21827, 27001, and 27004, the CMMI Measurement & Analysis Process Area, and the CMMI GQ(I)M template (provided as an appendix).

The main organizing principle of the Practical Measurement Framework addresses concerns of poor-quality, unreliable, and non-secure software through the measurement of software assurance goals and objectives at project, program, and enterprise levels. It presents generic key measures from the supplier, acquirer, and the practitioner perspectives and cross-references these measures to the resources mentioned above when appropriate. Each perspective identifies the Measure, Information Need, and Benefit grouped by activity. For example, Table 3 shows a (draft) portion of the document regarding Supplier Measures During Design.

Table 3. Supplier Measures During Design

	Measures	Information Need	Benefit
Design	Number of entry points for a module (should be as low as appropriate)	Reduce opportunity for back doors	Ascertain that future application handles data inputs as required Reduce opportunity for exploits Reduce attack surface
	Percent of data input components that positively validate all data input	Determine if data validation is handled as required	
	Percent of data input components that positively validate all data input	Identify origins of defects (injection points during the SDLC)	

TOOLS

Several tools now exist for checking source code for security vulnerabilities and often output measurements as explicit results. Although many companies delineate the conceptual basis for their tools, few offer specific guidance regarding the measurements employed. Two companies provide concrete examples of their use of measurement.

1. Microsoft's Secure Windows Initiative used the Relative Attack Surface Quotient (RASQ) as initially presented by Michael Howard [Howard 2003]. This calculated number is put forth as a cyclomatic complexity measure for security that yields a relative metric of a product's "attackability." The measure is based on the identification of all the external exposures in the product code, with the goal of reducing the product's attack profile. It is of limited use because the measures are meaningful only for like products, but an independent evaluation did confirm the measure's effectiveness [Ernst & Young LLP 2003]. Manadhata and Wing from Carnegie Mellon's Computer Science Department also successfully applied the measurement to Linux [Manadhata 2004].
2. Many static analysis tools exist that do a competent job of identifying common vulnerabilities as the code is being written by using contextual analysis that is language specific. For example, Ounce Labs' Prexis computes a V-density measure to relate the number and criticality of vulnerabilities in the code for project decision-making. As an integrated software risk management and vulnerability assessment product, Prexis includes (1) Prexis/Engine: Source Code Vulnerability Scanning and Knowledgebase Core, (2) Management Risk Dashboard, and (3) Developer Remediation Workbench for the product development life cycle.

Also see the following BSI content areas: Black Box Testing Tools, Code Analysis Tools, and Modeling Tools. Note that spreadsheet programs, statistical packages, and database programs can be very helpful for some measurement and analysis purposes. Some vendors also offer tools that harvest data from other databases and repositories to produce a variety of measurement reports.

Various development tools now include static and dynamic capabilities for analyzing security characteristics within the code, and integrated developer environments become easier to use as new releases cater to the community's need for better security tools. Several companies now offer complete development environments which incorporate functional, non-functional, and runtime analysis. White box testing tools are available to integrate into the development environment that offer interactive feedback and remediation suggestions to the develop-

er during the coding process. Developers can then see the impact of their coding decisions and can choose between suggested remedies to the code. This proactive ability of some of the new tools should not only help to accomplish secure coding but also improve quality.

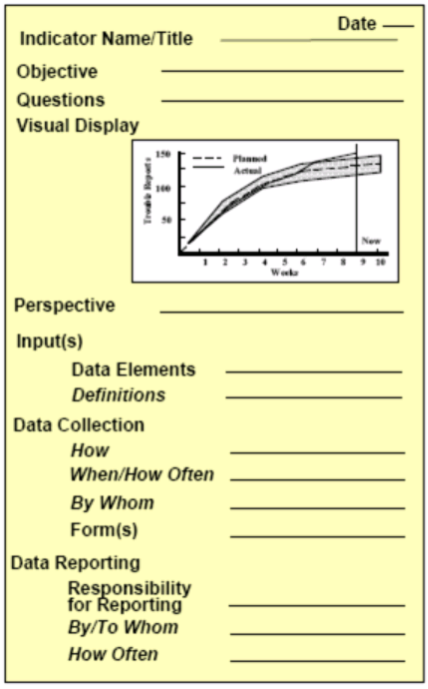
MATURITY OF PRACTICE

Software measurement is becoming a somewhat mature field, as evidenced by professional and international standards, specialized conferences, and several decades of literature and research. In spite of this history, the practice of software measurement is still highly variable among software development organizations, with many doing little to measure their projects and products during development. Also, very few organizations employ any form of measurement, much less sophisticated data analysis techniques for decision making, to assess the security characteristics of their products in a quantitative manner during development. Indeed, few even address security concerns in any manner. Very little exists in the published literature concerning the use of software measurement with respect to characterizing security concerns during software development.

APPENDIX: INDICATOR TEMPLATE

Current Version

The current version of the indicator template is shown below. Fields that have been added based on user feedback are shown in italics.



Field Descriptions

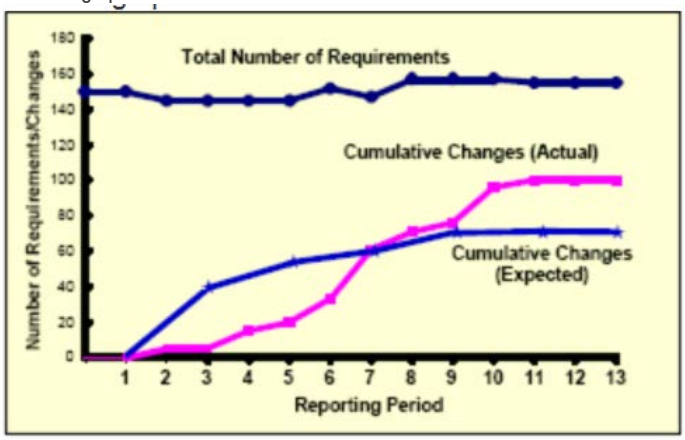
Date

Indicator Name/
Title

Objective — Describe the objective or purpose of the indicator.

Questions — List the question(s) the user of the indicator is trying to answer. Examples: Is the project on schedule? Is the product ready to ship? Should we invest in moving more software organizations to CMM maturity level 3?

Visual Display	Provide a graphical view of the indicator.
Perspective	Describe the audience (for whom is this display intended) for the visual display.
Input(s)	
- Data Elements	List all the data elements in the production of the indicator.
- Definition	Precisely define the data element or point to where the definition can be found.
Data Collection	
- How	Describe how the data will be collected.
- When/How Often	Describe when the data will be collected and how often.
- By Whom	Specify who will collect the data (an individual, office, etc.)
- Form(s)	Refer to any standard forms for data collection (if applicable) and provide information about where to obtain them.
Data Reporting	
- Responsibility for Reporting	Indicate who has responsibility for reporting the data
- By/To Whom	Indicate who will do the reporting and to whom the report is going. This may be an individual or an organizational entity.
- How Often	Specify how often the data will be reported (daily, weekly, monthly, as required, etc.)



Data Storage	
- Where	Indicate where the data is to be stored.
- How	Indicate the storage media, procedures, and tools for configuration control.
- Security	Specify how access to this data will be controlled.
Algorithm	Specify the algorithm or formula required to combine data elements to create input values for the indicator. It may be very simple, such as Input1/Input2, or it may be much more complex. It should also include how the data is plotted on the graph.
Assumptions	Identify any assumptions about the organization, its processes, life cycle models, and so on that are important conditions for collecting and using this indicator.
Interpretation	Describe what different values of the indicator mean. Make it clear how the indicator answers the "Questions" section above. Provide any important cautions about how the data could be misinterpreted and measures to take to avoid misinterpretation.
Probing Questions	List questions that delve into the possible reasons for the value of an indicator, whether performance is meeting expectations or whether appropriate action is being taken.
Analysis	Specify what type of analysis can be done with the information.
Evolution	Specify how the indicator can be improved over time, especially as more historical data accumulates (e.g., by comparing projects using new processes, tools, environments with a baseline; using baseline data to establish control limits around some anticipated value based on project characteristics).
Feedback Guidelines	Include a description of the procedure to use when recommending modification to the indicator template.
X-References	If the values of other defined indicators influence the appropriate interpretation of the current indicator, refer to them here.

Source: Goethert, Wolfhart & Siviy, Jeannine. Applications of the Indicator Template for Measurement and Analysis (CMU/SEI-2004-TN-024). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tn024.html.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120