**Practical Software and Systems Measurement Continuous Iterative Development**

**Measurement Framework**

**Part 2: Measurement Specifications: Automated Test Coverage**

Version 2.1

April 15, 2021

|  |  |  |
| --- | --- | --- |
| **Developed and Published by Members of:** | | |
| Practical Software & Systems Measurement | National Defense Industrial Association | International Council on Systems Engineering |
|  | NDIA 100 Year logo |  |
| Product No.  PSM-2021-03-001 |  | Product No.  INCOSE-TP-2020-001-06 |

|  |  |  |
| --- | --- | --- |
| **Editors:** | | |
| **Cheryl L. Jones**  US Army  [cheryl.l.jones128.civ@mail.mil](mailto:cheryl.l.jones128.civ@mail.mil) |  | **Geoff Draper**  L3Harris Technologies  [geoff.draper@l3harris.com](mailto:geoff.draper@l3harris.com) |
| **Bill Golaz**  Lockheed Martin  [willliam.h.golaz@lmco.com](mailto:willliam.h.golaz@lmco.com) |  | **Paul Janusz**  US Army  [paul.e.janusz.civ@mail.mil](mailto:paul.e.janusz.civ@mail.mil) |

Unclassified: Distribution Statement A: Approved for Public Release; Distribution is Unlimited

PSM Product Number: PSM-2020-06-001

INCOSE Product Number: INCOSE-TP-2020-001-06

**Copyright Notice:**

For this document, each of the collaborative organizations listed on the cover page is the sole manager of their products and services and are the only parties authorized to modify them. Since this is a collaborative product, modifications are managed through the participation of all parties.

General Use: Permission to reproduce, use this document or parts thereof, and to prepare derivative works from this document is granted, with attribution to PSM, NDIA, and INCOSE, and the original author(s), provided this copyright notice is included with all reproductions and derivative works.

Supplemental Materials: Additional materials may be added for tailoring or supplemental purposes if the material developed separately is clearly indicated. A courtesy copy of additional materials shall be forwarded to PSM ([psm@psmsc.com](mailto:psm@psmsc.com), attention: Cheryl Jones). The supplemental materials will remain the property of the author(s) and will not be distributed, but will be coordinated with the other collaboration parties.

Author Use: Authors have full rights to use their contributions with credit to the technical source.

Supplemental Notice from INCOSE: This work is an Affiliate Product per INCOSE Policy TEC-107 INCOSE Technical Product Development & Commercialization (26 October 2018). It is a technical product developed outside the INCOSE product development process and was made by INCOSE members in cooperation with PSM and NDIA; then approved by INCOSE to be distributed from INCOSE central channels. The authors own the copyright and take primary responsibility for proper branding, intellectual property, content quality and appropriate citations with INCOSE oversight based on this policy & related procedure.

Contents

[8. Measurement Specifications 1](#_Toc70227503)

[8.1 Automated Test Coverage (Product or Enterprise Measure) 1](#_Toc70227504)

List of Figures

[Figure 1: Automated Test Coverage (Project Level) 2](#_Toc70227505)

[Figure 2: Automated Test Pass/Fail Status 3](#_Toc70227506)

[Figure 3: Code Coverage from Automated Testing 4](#_Toc70227507)

[Figure 4: Automated Test Coverage (Enterprise Level) 5](#_Toc70227508)

# Measurement Specifications

## Automated Test Coverage (Product or Enterprise Measure)

|  |  |
| --- | --- |
| **Measure Introduction** | |
| **Description** | In an iterative development approach, it is important not only to efficiently verify new features but to ensure prior functionality is not impacted. Doing so manually can be time-consuming. Typically, code coverage is verified primarily in structural (white box) testing at the unit level, and requirements are verified primarily in functional/system test. Efficiency and throughput can be enabled by automated test suites executed at multiple levels (unit level, functional level, regression testing).  The extent to which automated testing is implemented is a business decision depending on objectives and constraints, such as velocity, quality, and cost vs. benefit. It may not be feasible or desirable to automate all testing. Projects may set planned test automation objectives, such as 70%-80% coverage based on their cost benefit analysis.  Often these automated test suites are integrated directly in the code pipeline and invoked upon each code commit and build, or in nightly regression test batch jobs. (Refer to Figure 2 for context.) Test results (tests passed, tests failed) can be distributed automatically in email so anomalies impacting the code quality and pipeline can be quickly identified and resolved. |
| **Relevant Terminology** | |  |  | | --- | --- | | Functional Testing | Testing against the requirements or function of the software, without considering the internal implementation. Sometimes termed black box testing. | | Structural Testing | Testing the internal structure, design, implementation, or logic of software, such as paths, conditionals, or branches through the code. Sometime termed white box testing. | |

|  |  |
| --- | --- |
| **Information Need and Measure Description** | |
| **Information Need** | How much of the testing is automated? How many tests have been validated and approved?  How much credit is given in formal test (e.g., DT/OT) for automated test? |
| **Base Measure 1** | Total Requirements *[integer > 0]* |
| **Base Measure 2** | Requirements Tested *[integer > 0]* |
| **Base Measure 3** | Requirements Tested Through Automation *[integer > 0]* |
| **Base Measure 4** | Requirements Tested Manually *[integer > 0]* |
| **Base Measure 5** | Code Constructs (e.g., classes, conditionals, files, lines, packages) *[integer > 0]* |
| **Base Measure 6** | Code Constructs Tested by Automated Test *[integer > 0]* |
| **Base Measure 7** | Automated Test Cases Passed *[integer > 0]* |
| **Base Measure 8** | Automate Test Cases Failed *[integer > 0]* |
| **Derived Measure 1** | Requirements Not Tested = (Total Requirements) – (Requirements Tested Through Automation) – (Requirements Tested Manually) *[integer > 0]* |
| **Derived Measure 2** | Percentage Requirements Tested Through Automation =  (Requirements Tested Through Automation) / (Total Requirements) \* 100 *[percentage]* |
| **Derived Measure 3** | Percentage Requirements Tested Manually = (Requirements Tested Manually) / (total requirements) \* 100 *[percentage]* |
| **Derived Measure 4** | Percentage Requirements Not Tested =  (Requirements Tested Not Tested) / (total requirements) \* 100 *[percentage]* |
| **Derived Measure 5** | Percentage Code Constructs Tested = (Code Constructs Tested by Automated Test) / (Code Constructs) \* 100 [percentage] *(for each code construct) [percentage]* |

|  |  |
| --- | --- |
| **Indicator Specification** | |
| **Indicator Description and Sample** | Figure 1 depicts the percentage of project requirements that are verified by automated vs. manual testing over time. In this example, the project set a planned objective for 70% automation, and ultimately met and exceeded that objective. Percentages are used rather than absolute values to facilitate comparisons across projects. The total number of requirements changes over time as more requirements are developed, and are plotted on the secondary axis to enable consideration of the scale and complexity of the test automation effort. Tradeoff decisions can be made on the benefit of investing further program effort to develop new automated test cases to increase coverage. This may include estimating the net impact on program throughput, quality, or cost.    Figure 1: Automated Test Coverage (Project Level)  At project startup an initial requirement set is established that evolves iteratively (with additions, modifications, deletions) across the project life based on collaboration with the product owner and other stakeholders. Test cases (automated and manual) are developed to verify requirements as they are implemented. By iteration 9, the automated test suite is verifying over 70% of requirements, supplemented by manual test cases that verify nearly all project requirements. In iteration 18, the product owner deleted a capability from the backlog and requirements count was reduced. Over time, additional automated tests are developed that increase automated coverage while reducing the dependence on manual testing, although both are supplemented regularly as new requirements are added. The project has sustained its automated test suite to generally meet the project objective of 70%-80% automated test coverage. |
|  | Effectiveness of automated testing should be monitored. The pass/fail success status of automated tests is often available from automated test tools, as illustrated below in Figure 2, so anomalies breaking the code pipeline can be quickly detected and resolved. The quantity of requirements covered in automating testing is depicted in the amplitude. Requirements that failed an automated functional test are shown in red, indicating quality of the pipeline over time. Some tools may also provide additional information, such as requirements that were skipped, or the requirements with no automated test.    Figure 2: Automated Test Pass/Fail Status  This automated report from the program test tool indicates a low number of requirements (<5) over time that failed automated testing. All test failures are investigated. Some of the test failures are due to enhancing the automated test scripts to verify new requirements as they are added, others are the result of regression test failures where baseline product functionality was impacted by new enhancements, but this quickly stabilizes as the product development baseline matures. |
|  | The extent of code structural coverage from automated (white box) testing can increase confidence in development baseline quality. In Figure 3 test coverage is collected for each increment and depicted by trends for % coverage of structural code constructs (classes, conditionals, files, lines, packages). The extent of coverage can indicate the risk or confidence in code quality, suggest a need for additional testing, or the potential risk of incurring defect escapes.    Figure 3: Code Coverage from Automated Testing  100% of packages and 95% of classes are addressed by automated tests. 85% of the code (lines of code) and 75% of branches are currently exercised; coverage dropped in iteration 980 (to 70% of code, 65% of branches) as new functionality was added, but has continued to grow in subsequent releases as the automated test suite was expanded to address these enhancements. The project has set a target for > 80% of code and branches exercised in automated testing, so the test suite is being enhanced for additional logic test cases focusing on the most risky or complex modules. |
| **Indicator Description and Sample**  **(continued)** | At the enterprise level, the extent of automated testing utilized across projects can be monitored, as reflected in Figure 4. The enterprise may set business objectives for the extent of automated testing across projects (e.g., 70%), subject to project-specific characteristics and constraints.    Figure 4: Automated Test Coverage (Enterprise Level)  Automated test coverage percentages are collected from projects and aggregated at the enterprise level to monitor the success of implementing automated testing. Measures are displayed for each project in both relative (%) and absolute terms (Requirements Verified). Absolute values are used for context in evaluating the overall impact of the project automated test coverage; larger projects may have greater challenges in scope but also more resources available to realize the benefits of automation. Some projects are early in their development cycle and development of automated test cases are still in work. Overall, the project average is 68% automation, but when weighted by the number of requirements verified the coverage is 73% due to the higher impact from larger projects. Analysis and actions at the organizational level will depend on the characteristics of the individual projects, the extent to which performance and quality measures are impacting objectives, and the extent to which they may be positively impacted by investing in additional automation. |
| **Analysis**  **Model** | Automated Test Coverage (Project Level):   * What percentage of functional requirements are verified with automated testing? Is each requirement fully covered by the automated testing, or are some aspects not verified? * Any requirements not verified automatically must be verified manually, which can impact productivity, schedule, and resources. Apply decision tradeoffs for the cost vs. performance benefit of investing effort to expand the extent of automated test coverage.   Automated Test Pass/Fail Status:   * Are automated tests completing successfully, or are there anomalies impacting the code pipeline that should be investigated? * Automated tests are typically conducted regularly as part of the code and unit tests in the code development pipeline, such as upon each code commit or in nightly regression tests. Summary test reports can be automatically generated and distributed by the automated test tools. 100% success of automated tests passing is often a criterion for advancing the code baseline to production. Discrepancies could be in the code, or in the test cases themselves, but either should be investigated.   Code Coverage from Automated Testing:   * How much of the code structure is covered by the automated test suite? Which parts of the code are not covered (e.g., any safety critical code, interfaces, interoperability requirements)? * Code coverage is a tradeoff between investment, risk, and return; although 100% coverage may be desirable, that might not be practical within available environments, resources, interfaces, and constraints.   Automated Test Coverage (Enterprise Level):   * What is the extent of automated testing conducted across the organization’s projects? What benefits to organizational performance (e.g., cycle time, quality, throughput) are enabled by effective automated testing?   Automated testing is a primary enabler for achieving efficiency, quality, and cost savings at both the project and organizational levels. Organizations should monitor automated test measures in relation to achievement of their desired performance objectives. |
| **Decision Criteria** | Automated test coverage alone is not an objective; it is the associated gains in accelerating performance and improving product quality at the project and organizational levels that make investments in automation worthwhile. Automation measures should be evaluated in the context of other performance measures, such as those defined elsewhere in the PSM CID measurement framework. Industry experience suggests that automation in the range of 70%-80% is often beneficial in producing improved performance outcomes, but this may vary by domain or application.  If automation measures are lower than planned, or if there are process effectiveness or product quality issues that are impacting objectives, consider root cause analysis and decision tradeoffs to assess the impact and determine if they can be improved by further investments in test automation. |

|  |  |
| --- | --- |
| **Additional Information** | |
| **Additional Analysis Guidance** | Test automation and coverage are key elements of achieving faster and more comprehensive releases with higher code quality. These should be used in conjunction with quality measures to ensure the adequacy of testing and achieve acceptable, inherent quality levels. A reasonable goal is to achieve near instantaneous automated test results with acceptable quality. Testing efficiency and speed are closely related to achieving other performance measurement objectives such as lead time, cycle time, and release frequency. Robustness of the testing conducted should also be considered (e.g., stress testing, boundary conditions on valid data inputs).  Additional project performance measures, such as effort, schedule, and cost, can be correlated with automated test coverage measures to evaluate the performance benefits (e.g., cost savings, productivity, quality) achieved through automated testing.  Alternative thresholds or weighting could be applied to automated test coverage scores based on characteristics of a project or component, such as size, complexity, reuse, criticality, or other dependencies. |
| **Implementation Considerations** | Measures for code coverage and requirements coverage are directly available from many automated development tools commonly integrated across the tool chain. However, the emphasis should be on thorough testing sufficient to ensure product quality rather than achieving high code coverage numbers. Code coverage is an important factor, but by itself, is not sufficient to ensure product quality. Automated test cases could focus on areas of high risk, complexity, or dependencies where repeatability or regression testing are important factors, especially in the near term.  Relying solely on automated test tools and scripts may not be wholly sufficient to exercise all functionality needed (e.g., user interfaces, databases). It may be necessary to supplement automated test scripts with manual effort to execute additional test cases and validate that the automated test is sufficiently representative of the overall functionality.  Automated testing may be conducted at various or multiple points in the workflow, for instance before or after the baseline merge. A best practice is to execute automated test suites nightly or as part of the pipeline following each code commit.  For existing systems, the enterprise will need to make a business decision as to whether it is worth the investment to develop automated tests. This will be dependent on the necessary infrastructure to support automated test, the expected lifecycle of the system, the level of updates/regression test typically required, etc.  Automated test scripts are a valuable work asset that should be sustained in a manner similar to source code. Test scripts may need to be enhanced or refactored as the product evolves. |

|  |  |
| --- | --- |
| **Additional Specification Information** | |
| **Information Category** | Process Performance (Process Effectiveness) |
| **Measurable Concept** | Process Effectiveness |
| **Relevant Entities** | System, Test cases |
| **Attributes** | Amount tested, amount automated tested |
| **Data Collection Procedure** | Data is typically collected by automated tools upon execution of test scripts as part of standard pipeline workflows. Results are recorded in team tracking tools. Summaries of test results and coverage can often be provided automatically nightly or upon completion. |
| **Data Analysis Procedure** | Data is reviewed and analyzed to ensure adequate quality for each candidate product. Discrepancies in process effectiveness, product quality, or test coverage not meeting threshold targets may indicate updates to code or test scripts are necessary. |