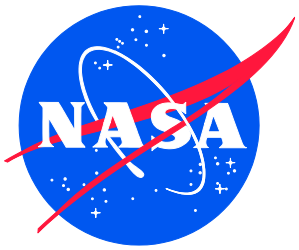
NASA/TM–20205011566
NESC-RP-20-01515



Cyclomatic Complexity and Basis Path Testing Study

Michael D. Squire/NESC
Langley Research Center, Hampton, Virginia

Laura A. Maynard-Nelson
Glenn Research Center, Cleveland, Ohio

Terry A. Brown
Marshall Space Flight Center, Huntsville, Alabama

Robert T. Crumbley
NASA Headquarters, Washington D. C.

Gerald J. Holzmann
Jet Propulsion Laboratory, Pasadena, California

Michael Jennings
U. S. Airforce, Tinker Air Force Base, Oklahoma

Kequan Luu, and Walter F. Moleski
Goddard Space Flight Center, Beltsville, Maryland

Jay D. Marchetti
Carnegie Mellon University–Software Engineering Institute, Pittsburgh, Pennsylvania

December 2020

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

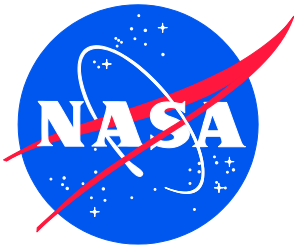
For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/
and select the "General" help request type.

NASA/TM–20205011566
NESC-RP-20-01515



Cyclomatic Complexity and Basis Path Testing Study

Michael D. Squire/NESC
Langley Research Center, Hampton, Virginia

Laura A. Maynard-Nelson
Glenn Research Center, Cleveland, Ohio

Terry A. Brown
Marshall Space Flight Center, Huntsville, Alabama

Robert T. Crumbley
NASA Headquarters, Washington D. C.

Gerald J. Holzmann
Jet Propulsion Laboratory, Pasadena, California

Michael Jennings
U. S. Airforce, Tinker Air Force Base, Oklahoma

Kequan Luu, and Walter F. Moleski
Goddard Space Flight Center, Beltsville, Maryland

Jay D. Marchetti
Carnegie Mellon University–Software Engineering Institute, Pittsburgh, Pennsylvania

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

December 2020

Acknowledgments

The assessment team would like to acknowledge the contributions of both the Core Flight Software (CFS) and the Autonomous Power Controller (APC) Teams in providing data on complexity and testing. The team would also like to thank the following peer reviewers: Tim Barth, Dan Dorney, Caren Ensey, Steve Gentz, Pam Pittman, and Lorraine Prokop.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

NASA Engineering and Safety Center
Technical Assessment Report

Cyclomatic Complexity and Basis Path Testing Study

November 16, 2020

Report Approval and Revision History

NOTE: This document was approved at the November 16, 2020, NRB. This document was submitted to the NESC Director on November 17, 2020, for configuration control.

Approved:	*Original Signature on File*	11/18/20
	NESC Director	Date

Version	Description of Revision	Office of Primary Responsibility	Effective Date
1.0	Initial Release	Michael D. Squire NESC Principal Engineer, LaRC	11/16/2020

Table of Contents

List of Figures

List of Tables

Technical Assessment Report

1.0 Notification and Authorization

Mr. Ralph Roe, NASA Chief Engineer, requested the NASA Engineering and Safety Center (NESC) to conduct a study to determine the benefits of cyclomatic complexity and basis path testing (BPT) for software and whether they should be required. The principal focus of the assessment was to assess the use of cyclomatic complexity and BPT on safety-critical software. The purpose was to ensure that safety-critical software is not overly complicated to the point of increasing coding errors and that verification is more robust than for non-safety-critical software.

Key stakeholders for this assessment included the NASA Chief Engineer, the Associate Administrator for the Human Exploration and Operations Mission Directorate (HEOMD), NASA's Commercial Crew Program, NASA's Software Engineering and Software Assurance communities, NASA program and project managers, and the Office of Safety and Mission Assurance (OSMA).

2.0 Signature Page

Submitted by:

Team Signature Page in File – 12/3/2020

Mr. Michael D. Squire Date

Significant Contributors:

Ms. Laura A. Maynard-Nelson Date

Mr. Terry A. Brown Date

Mr. Robert T. Crumbley Date

Mr. Gerard J. Holzmann Date

AG Michael Jennings Date

Mr. Kequan Luu Date

Mr. Jay D. Marchetti Date

Mr. Walter F. Moleski Date

Mr. Michael A. Riley Date

3.0 Team List

Name	Discipline	Organization
Core Team		
Michael Squire	NESC Lead	NESC
Laura Maynard-Nelson	Technical Lead	GRC
Tim Crumbley	OSMA	HQ
Terry Brown	Software Engineer	MSFC
Gerard Holzmann	Software Engineer	Nimble Research/JPL
Michael Jennings	Software Sustainment Senior Lead	U.S. Air Force
Suzanne Knispel	Software Testing Engineer	Boeing
Kequan Luu	Software Engineer	GSFC
Jay Marchetti	Software Engineer	CMU–SEI
Walter Moleski	Software Testing Engineer	GSFC
Mike Riley	Software Engineer	CMU–SEI
Consultants		
Lorraine Prokop	NASA Technical Fellow for Software	JSC
Business Management		
Tejal Fairfield	Program Analyst	LaRC/MTSO
Assessment Support		
Jocelyn Santos	Project Coordinator	LaRC/AMA
Tejal Fairfield	Planning and Control Analyst	LaRC/AMA
Jenny DeVasher	Technical Editor	LaRC/AS&M

3.1 Acknowledgments

The assessment team would like to acknowledge the contributions of both the Core Flight Software (CFS) and the Autonomous Power Controller (APC) Teams in providing data on complexity and testing. The team would also like to thank the following peer reviewers: Tim Barth, Dan Dorney, Caren Ensey, Steve Gentz, Pam Pittman, and Lorraine Prokop.

4.0 Executive Summary

After the software failures experienced during Boeing's Crew Space Transportation (CST)-100 Orbital Flight Test (OFT), NASA Chief Engineer Ralph Roe expressed a concern that NASA's software testing requirements were potentially insufficient. The head of the HEOMD and representatives from the Autonomous Power Controller (APC) System also expressed concern regarding the complexity of the software and the amount of testing required. The principal focus of the assessment was to assess the use of cyclomatic complexity and basis path testing (BPT) on safety-critical software. The purpose was to consider adding requirements for complexity, and testing based on complexity metrics, to NASA's software standards for safety-critical software with the overall objective of reducing errors.

The current version of the NASA Procedural Requirements (NPR) for Software[1] contains requirements for unit-level testing and the use of static analysis tools. However, the requirements do not state what type of unit testing is required, nor which items should be considered in static analysis tool results. These decisions are delegated to programs/projects to determine. This version of the NPR has no requirement regarding software complexity, whether safety-critical or not. The updated NASA Standard for Software Assurance and Software Safety[2] contains a requirement for assessing the cyclomatic complexity for all safety-critical code modules. This standard sets the cyclomatic complexity level for safety-critical functions at 15 or lower.

An assessment team comprising NASA software engineers, industry partners, military personnel, and academia was formed to review the use of cyclomatic complexity and BPT and whether they should be applied to NASA projects and programs. The team assessed the current use of these techniques at NASA Centers and other agencies and companies. The team examined some software products for existing projects to evaluate their complexity levels and the types and amount of testing completed. The team also considered tools that could help with determining complexity and researched previous software failures and their causes (i.e., whether complexity played a factor).

Assessment team members provided information on the types of software testing available at different levels (e.g., unit, configuration item, subsystem, system) and evaluated the types in use. Typically, the Agency's approach to testing involves some unit level testing, continuous integration testing, and requirements testing.

Throughout this investigation, while focused on cyclomatic complexity and BPT, the assessment team did not limit itself to those areas. Other data and factors were also evaluated, including requirements development and code coverage requirements. The team members agreed that a maximum value of 15 should be required when assessing the cyclomatic complexity for safety-critical software. However, evidence showed that limiting complexity alone would not guarantee software less prone to errors and failures. The point of the requirement is to minimize risk, minimize testing, and increase reliability associated with safety-critical software code components, thus reducing the chance of software failure resulting in a major mishap.

For the testing method portion of this effort, the assessment team determined that while BPT would be beneficial, modified condition/decision coverage (MC/DC) would be a more robust

[1] NPR 7150.2C, NASA Software Engineering Requirements

[2] NASA-STD-8739.8A, Software Assurance and Software Safety Standard

testing approach. However, MC/DC is likely more taxing to programs/projects in terms of cost and schedule. Complete test coverage for software safety-critical code should be required. The concept of using untested code in a hazardous condition should not be considered acceptable. The updated NASA Standard for Software Assurance and Software Safety requires confirmation that 100% code test coverage has been achieved or addressed for all identified software safety-critical components or provide a risk assessment explaining why the test coverage is not possible for the safety-critical code component. If safety-critical code has not been tested, the program/project should understand why and discuss the risk associated with the hazard activity and the untested code.

The NESC recommends programs/projects use the MC/DC approach, a code coverage criterion commonly used in software testing. MC/DC is similar to condition coverage, but requires testing every condition in a decision independently to reach full coverage. The approach means that each condition must be executed twice, with the results true and false, but with no difference in the truth values of all other conditions in the decision. Also, it must be shown that each condition independently affects the decision.

With this metric, some combinations of condition results are redundant and not counted in the coverage result. The coverage of software code is the number of executed statement blocks and non-redundant combinations of condition results divided by the number of statement blocks and required condition result combinations.

Code coverage is a way of measuring the effectiveness of test cases. The higher the percentage of code covered by testing, the less likely it is to contain errors when compared to code with a lower coverage score. Three other types of code coverage are worth considering with MC/DC: statement coverage (SC), decision coverage (DC), and multiple condition coverage (MCC).

The NESC recommends the following two requirements be added to NPR 7150.2:

3.7.4 The project manager shall ensure that there is 100% code test coverage using the MC/DC criterion for all identified safety-critical software. (SWE-208)

Aerospace and space guidance prioritizes safety in the software development life cycle (SDLC). MC/DC represents a compromise that finds a balance between rigor and effort, positioning itself between DC and MCC. MC/DC requires a smaller number of test cases in comparison to MCC, while retaining a high error-detection probability.

3.7.5 The project manager shall ensure all identified safety-critical software components have a cyclomatic complexity value of 15 or lower for each software component. (SWE-209)

Cyclomatic complexity is a software metric used to measure code complexity. These metrics measure independent paths through source code. The point of the requirement is to minimize risk, minimize testing, and increase reliability associated with safety-critical software code components. The developer should assess all software safety-critical components with a cyclomatic complexity score over 15 for testability, maintainability, and code quality.

The assessment team members felt it was important to understand that while these proposed requirements will help to create better software systems, they are not the only areas to consider. The team provided a list of factors they consider critical to a robust software system, including architectural complexity, requirements analysis, complete verification approach, and independent code reviews. This assessment was narrowly focused; a broader assessment of the overall approach to software development may be warranted.

5.0 Assessment Plan

While not required, an informal plan was developed as a starting point for the team. Assessment team members reviewed the approaches being followed across the software discipline internal to NASA, and across industry, the military, and academia. Table 5-1 lists the external (to NASA) standards and guidelines that were discussed. The assessment team also spoke to software personnel with experience from Blue Origin, AT&T Bell Labs, Google, Uber, and Codemanship (a software consulting firm). Also considered were what tools are available to help determine cyclomatic complexity for different applications and were shown demonstrations of some of those tools. The team reviewed a limited number of software sets, but enough to understand how complex some software systems are and how effective BPT can be. Team members interviewed software developers who have employed various software testing techniques to gauge what the best approach for safety-critical software might be. Finally, the team assessed problems and errors in software development and previous space flight missions to understand what areas of software development might have benefited from more rigor.

Table 5-1. External NASA Documents Consulted

Safety Design Criteria for Nuclear Weapons Systems Software, USAF 91-119
Medical Device Software Life Cycle Process, IEC62304
Motor Industry Software Reliability Association (MISRA) Rule Set, 1997
MISRA Rule Set, 2004
Joint Software System Safety Committee Software System Safety Handbook, 1999
European Space Agency Coding Rules, 2000
Goddard Flight Software Branch Coding Rules, 2000
Mars Reconnaissance Orbiter Coding Rules, 2002
JPL Multi-Mission System Architecture Platform Coding Rules, 2005
Joint Strike Fighter Air Vehicle Rules, Rev. C, 2005
JPL Space Interferometry Mission Realtime Control Subsystem Coding Rules, 2005
JPL Mars Science Laboratory Coding Rules, 2006
JPL Power of Ten Rules, 2006
JPL Institutional Coding Standard, 2008

6.0 Problem Description and Assessment Team Evaluation

Software systems have become more complex over the past several years while becoming increasingly responsible for running critical spacecraft systems. It is vital that these software systems are fully verified to ensure the safety of crew members and vehicles. As seen during the CST-100 OFT flight, insufficient decision-testing coverage may have contributed to potentially catastrophic errors going undetected until flight. The more complex the code, the more difficult it is to adequately test and verify.

The complexity of a software-intensive system expresses itself in two primary ways: architectural complexity and structural, or source code, complexity. Software architecture is the fundamental organization of a system, as embodied in its components and their relationships to each other and the environment. All software systems have an architecture, whether known and documented or not. Coupling and cohesiveness among a system's components and subsystems reflect the dependency view of its architecture. Both the number and types of interdependencies

among the constituent elements of a system indicate the level of the system's architectural complexity. Systems containing source files that are highly interdependent or have numerous bidirectional or cyclic dependencies in effect encode system information, and hence complexity, through their dependencies. Such systems are often referred to as "brittle" to indicate that they are difficult to test, reuse, and maintain over the SDLC. Certain architectural patterns—for example, layering—are often used by software architects as mitigations against otherwise unconstrained dependencies and the increased software architectural complexity they precipitate.

Alternatively, the complexity within the source code of a single module, function, or block is structural complexity. Several metrics correlated to structural code complexity are in use and supported by various code analysis tools. These include source lines of code (SLOC), function points, and Halstead's complexity measures. However, the most common structural complexity metric is cyclomatic complexity [ref.11], which measures the number of decisions within the code based upon its control flow graph. Cyclomatic complexity is defined as equal to the number of test cases required to test all linearly independent, or "basis," paths through the code (see Figure 6-1). Hence, BPT, one form of structured, or "white-box," testing, is driven by the code's cyclomatic complexity and has become a commonly used criterion for measuring software testing coverage. To be "linearly independent" means that each path has at least one edge that is not in one of the other paths. For instance, if the source code contained no control flow statements (i.e., conditionals or decision points), the cyclomatic complexity would be one, since there would be only a single path through the code. If the code had one single-condition IF statement, there would be two paths through the code: one where the IF statement evaluates TRUE and another where it evaluates to FALSE, so the cyclomatic complexity in this case is two. Two nested single-condition IFs, or one IF with two conditions, would produce a cyclomatic complexity of three. Figure 6-1 shows a graphical representation of a function with a cyclomatic complexity of three. It is important to note that the cyclomatic complexity is not tied directly to the number of SLOC. A function that runs start to finish with no options has a cyclomatic complexity of one, regardless of the number of lines.

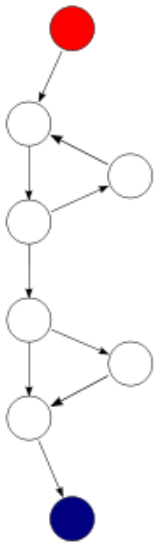


Figure 6-1. Graphical Representation of Function with Cyclomatic Complexity of Three

There are different types of cyclomatic complexity. For example, standard cyclomatic complexity (CC or CC1) is equal to the number of decisions + 1 for the code in question, as

described above. Strict cyclomatic complexity (CC2) [see ref. 11] adds one for each Boolean condition within a compound decision predicate, rendering CC2 difficult to "game" (i.e., lowering the measured complexity through moving otherwise nested code decision blocks into complex multi-condition decision logic). Another variation, modified cyclomatic complexity (CC3), reduces the penalty on multi-way decision branches, such as switch() statements in the C language, by counting them as one decision. CC3 may be sensible for certain non-safety-critical application types that use event handlers or finite state machines heavily. It is important to understand and choose which type of cyclomatic complexity will be used, as many tools support one or more and their specific terminology may differ from that described here.

Throughout NASA, cyclomatic complexity is used by some Centers, but is not universally required. Some projects are collecting the metric, but may or may not modify code based on its value. Notably, the two largest NASA programs with safety-critical flight software (i.e., Space Launch Systems (SLS) and Multi-Purpose Crew Vehicle) determine cyclomatic complexity and assess code that exceeds defined thresholds to determine whether it is acceptable or needs to be modified. Other projects use the cyclomatic complexity number during peer reviews and inspections to increase understanding of the software. In all cases where cyclomatic complexity is used, rules have been established for handling cases where functions exceed the target value.

According to presentations the assessment team received from the U.S. Air Force, industry, and academia, cyclomatic complexity is not used universally, but those who use it find it helpful. In data collected from industry, there was no evidence that cyclomatic complexity played a dominant role in software development. However, it was one of several factors identified with code defect density, along with metrics describing others such as code churn (i.e., how often code is changed), assertion density (the percentage of code that performs self-checking functions, e.g., in the form of an assertion—a typical target for safety-critical code is an assertion density of 2% or higher), and inter-module dependencies. Incomplete or missing requirements and insufficient requirements traceability were also identified as causes for software problems, especially those found later in the mission life cycle. Several study papers reviewed by the assessment team discussed the use of cyclomatic complexity and other metrics in software development [refs. 1-6]. Scholarly data on code complexity metrics to reduce software errors are mixed because of the difficulty in performing controlled studies across different application areas, programming team skill levels, and software languages. Most general software coding and quality standards (e.g., ISO 25010, Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)) do not include cyclomatic complexity. However, some safety-critical coding standards do. One example is the Joint Strike Fighter (JSF) coding standard, which uses a cyclomatic complexity maximum of 20.

For the NASA Centers that use cyclomatic complexity, many projects with safety-critical functions used complexity levels of 20 or less. The Air Force uses a complexity level of 15-20, whereas the academia and industry examples the assessment team reviewed (e.g., MISRA, AUTOSAR, Hersteller Initiative Software (HIS), JSF AV++) used complexity requirements in the 10–20 range. Code with lower complexity also translates into software that is easier to maintain. The maintainability index for a given piece of software is a combination of SLOC, cyclomatic complexity, and Halstead Volume. Software with a low cyclomatic complexity requires less unit testing and validation and will allow for fewer significant architectural changes during the SDLC.

Based on this, the assessment team chose a maximum of 15 for safety-critical software, with the understanding that this may change in the future to reflect the state of Agency software development. The team did not reach a consensus on which version to require (i.e., CC1, CC2, or CC3), deferring to programs/projects to decide how strict or lenient they felt appropriate and assuming adequate rationale. It is important that NASA not use cyclomatic complexity alone, but rather with other metrics, such as architectural complexity, code and requirements changes, and defect density, when evaluating software for errors.

The second objective of this assessment involved the testing of complex software. Developers across the Agency typically focus software testing on meeting system-level requirements, but are not required to verify that every decision or branch is performed correctly at the unit level. Unit-level testing is required, but it is not always clear to what depth it should occur, since most of NASA's testing revolves around requirements verification and many developers are concerned only with meeting those requirements. Ideally, the primary objective of software testing should be meeting requirements to reduce risk, rather than prioritizing nontechnical aspects, such as style and formatting. The strategy of blindly meeting requirements without ensuring the requirements are correct has led to potentially mission-critical errors. Finally, every defect or anomaly discovered post-release reveals a flaw in testing and should be addressed through additional testing, requirements, or both.

Many types of software testing can be used for unit-level testing. BPT is tightly coupled with cyclomatic complexity, requiring more testing effort for code of higher complexity, which is itself an incentive for developers to keep their code below some reasonable threshold. MC/DC, being even more rigorous than BPT, is more robust and thus suitable for safety-critical software applications when used in conjunction with functional and requirements testing. It is important to understand that this type of testing needs to occur at the unit level and be developed along with the software code to be most effective and less costly if problems are found.

BPT, or structured testing, is a white-box method for designing test cases. The method analyzes the software control flow graph of a software program to find a set of linearly independent paths of execution. The method normally uses cyclomatic complexity to determine the number of linearly independent paths and generates test cases for each path. BPT guarantees complete branch coverage (i.e., all edges of the control flow graph), but achieves that without covering all possible paths. Figure 6-2 provides an example of a function with cyclomatic complexity and three paths for BPT; the fourth path, shown in yellow, is redundant.

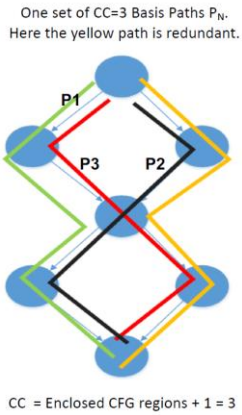


Figure 6-2. BPT Example

MC/DC is a code coverage criterion used in software testing that requires all of the following during testing:

- Each entry and exit point is invoked.
- Each decision takes every possible outcome.
- Each condition in a decision takes every possible outcome.
- Each condition in a decision is shown to independently affect the outcome of the decision.
- Independence of a condition is shown by proving only one condition changes at a time.

MC/DC is used in avionics software development guidance (DO-178B and DO-178C[3]) to ensure adequate testing of the most critical (Level A) software, (i.e., software that could provide or prevent failure of continued safe flight and landing of an aircraft). Table 6-1 shows an example of the use of MC/DC for testing a function of code.

Table 6-1. MC/DC Testing Example

Example: if (A && B && C) {statement block;}				
Test #	**A**	**B**	**C**	**Result**
7	1	1	1	1
6	1	1	0	0
5	1	0	1	0
4	1	0	0	0
3	0	1	1	0
2	0	1	0	0
1	0	0	1	0
0	0	0	0	0
To meet MC/DC decision coverage aspect: Perform tests #7 and #6				
To meet independent condition coverage:				
Variable A Perform tests 7 and 3		Variable B Perform tests 7 and 5		Variable C Perform tests 7 and 6
Result: Four test cases needed				

As the example shows, MC/DC provides a robust testing methodology because it ensures all decisions and paths are verified in a given function. The projects using cyclomatic complexity as a metric and containing safety-critical code are using MC/DC for their testing approach. Other standards, such as the FAA's DO-178C,[4] require full MC/DC test coverage for safety-critical avionics systems.

While SC and DC are subsets of MD/DC, MCC is actually similar to MC/DC, though it is more extreme. In MCC, all statements must be executed and all combinations of truth values in each decision must occur at least once to reach full coverage. In the Table 6-1 example, if a developer runs all seven test cases, they are performing MCC. While performing all seven test cases is more complete, there is no value added since the independent outcome is what really matters and that is covered by MC/DC.

[3] DO-178C, Software Considerations in Airborne Systems and Equipment Certification (replaced DO-178B in 2012)

[4] DO-178C, Software Considerations in Airborne Systems and Equipment Certification

7.0 Cyclomatic/Static Analysis Tools

Numerous available static analysis tools can calculate cyclomatic complexity for one or more source code files. Some tools, such as the Unified Code Counter (UCC) produced by the Center for Systems and Software Engineering at the University of Southern California, are freely available. UCC produces SLOC values and standard cyclomatic complexity (CC1) for each function or method in a multi-file source code system. Other commercial static analyzers, such as Scientific Toolworks' Understand, can identify all varieties of cyclomatic complexity (i.e., CC1, CC2, and CC3), the maximum function or method cyclomatic complexity for each file within the application, and visualizations of the code base that show the cyclomatic "hot spots" for drilling down via closer human analysis. Beyond these two tools, many other static analyzers and some integrated development environments (e.g., Atlasean's Bamboo) measure and display cyclomatic complexity for utilization by software developers, testers, and quality assurance analysts.

Tools are also available for assessing risk associated with cyclomatic complexity. One methodology is the Risk Management Framework, described in NIST 800-37 [ref. 10]. This standard uses six steps when accounting for risk: categorize risk, select controls, implement controls, assess controls, authorize their use, and monitor the implemented controls. This method of risk oversight helps ensure the associated design controls implemented to deal with risk are constantly monitored. If a design control becomes inadequate, constant monitoring will highlight additional control needs.

Similarly, dependency analysis for assessing architectural complexity is supported by commercial static analysis tools, such as Understand, Lattix Architect, and Silverthread. Lattix Architect and Silverthread use dependency structure matrix visualizations of software dependencies to show system coupling and cohesion, as well as to isolate cyclic dependencies within the codebase. These tools can also be used to enforce desired architectural design constraints on the system during the SDLC when run as part of a continuous integration stack. Some organizations that use the Silverthread tool also use the MITRE Code Assessment Toolset for automated static analysis. The key to these tools is that they are used frequently, starting in the development phase of the life cycle; based on a consistent set of rules and parameters; and automated so results can be compared on a regular basis.

The prevalence of MC/DC for testing safety-critical avionics software has led to MC/DC coverage support by a number of software test coverage tool vendors, including Rapita Systems, VectorCAST, LDRA, and Parasoft. These tools can assist software testers in creating test cases and calculating the resulting test coverage.

8.0 Data Analysis

This assessment team reviewed three software sets to understand complexity levels in flight software projects at this time: the SLS Flight Software, the Core Flight Software (CFS) bundle and applications, and the APC system. SLS Flight Software is a major in-house-built software system. The CFS bundle is in wide use and has been for several years, being added to by the open-source user community. The APC software is a Glenn Research Center in-house product in development, with interim deliveries to JSC for concept verification and potential use in future NASA missions.

For the SLS Flight Software, the applied coding standard required cyclomatic complexity of no higher than 20. For any units exceeding that level, either the code was reworked or a waiver was

processed. An exception to this rule was allowed for large switch statements. Cyclomatic complexity was calculated using the QAC++ tool from Programming Research Limited (PRQA, now owned by Perforce Software). NASA Independent Verification and Validation also calculated cyclomatic complexity using a different toolset and reported results to the SLS Flight Software Team. The results indicated that the average complexity of the SLS Flight Software code is ~2.9. Only one waiver was processed for a portion of legacy guidance, navigation, and control code.

For the CFS, 34 of 975 functions had a cyclomatic complexity level greater than 15. For the CFS applications, 111 of 2,426 functions exceeded that level. For the APC, it was 67 of 2,498. The interesting aspect is that without a requirement to do so, the CFS and APC teams kept their complexity levels low for the majority of their code. The CFS applications code was not written by one team, but by the CFS community, and with no guidance, fewer than 5% of the applications exceed a complexity of 15. As parts of the APC software were recoded to use the CFS bundle as their base, the complexity level of the functions decreased, with only 1 function of 113 exceeding the complexity level of 15.

The CFS and APC development teams were not required to follow either BPT or MC/DC testing methodologies. CFS followed the Goddard Open Learning Design Rules for Systems Testing and Requirements Verification. The developers followed their internal unit test standard and best practices for continuous integrated testing of the units and coverage testing. An independent team of software personnel tested all the requirements. The APC team had a bare minimum set of software unit level testing due to a limited team size.

The assessment team asked the APC developers to perform BPT on two or three functions, at least one with a high cyclomatic complexity value and one with a lower value. The results provided an evaluation of not just how difficult and time-consuming BPT is, but also where complexity levels become burdensome. The APC software lead chose three functions with cyclomatic complexity levels of 28, 9, and 5. The software lead documented how long it took to create a flow diagram, determine the test cases, and write the tests. For a complexity level of 28, these tasks took 31 hours, compared to less than one hour for the lowest complexity function level of 5. The software lead estimated his confidence level for each of these functions achieving full testing coverage. For the lower complexity levels, the estimate was close to 100% confident full coverage would be obtained, while for the higher complexity function, the estimated confidence level was 30%. This could be improved with independent verification that all paths were tested, but would increase the time and resources necessary to complete the testing.

The assessment team asked the APC developers to perform a similar activity for MC/DC. The developers used the same functions assessed for BPT, as well as an additional function with a complexity of 18. The developers indicated that the rules of MC/DC made it easier to create and evaluate the truth table associated with each function and determine which paths to test to achieve the desired results of testing full nominal and off-nominal functionality. The number of tests decreased for most functions, and the amount of time to develop those tests also decreased. It was also noted that the smaller the cyclomatic complexity, the smaller the difference in the number of tests for BPT and MC/DC. In some cases, the number was identical. Table 8-1 shows the difference in the number of test cases and the time it took to develop testing for different complexity (MCC) functions, using BPT and MC/DC. For the more complex functions, the time savings between BPT and MC/DC methodologies was 30 to 50%.

Table 8-1. Comparison of Test Cases and Time for BPT (top) and MC/DC (bottom)

FUNCTION	MCC	SLOC	BPT Tests	Time to Create Flow Diagram & Determine Test Case Paths	Time to Write Tests	Total Time (hours)
ServiceLoadShedMsm::HandleLoadShedGatewayPrm	28	108	28	16.000	15.000	31.000
ServiceEnergyAvailabilityPpe::InitSocProfiles	9	23	9	0.800	0.850	1.650
EnergyAvailableGatewayPrm::GetEnergyPerPhase	5	17	5	0.167	0.583	0.750
FUNCTION	**MCC**	**SLOC**	**MC/DC Tests**	**Est Time: Create Truth Table**	**Time to Write Tests**	**Total Time (hours)**
ServiceLoadShedMsm::HandleLoadShedGatewayPrm	28	108	17	1.000	3.000	4.000
ServiceEnergyAvailabilityPpe::InitSocProfiles	9	23	6	0.167	0.650	0.817
EnergyAvailableGatewayPrm::GetEnergyPerPhase	5	17	5	0.083	0.500	0.583
PowerSystem::GetSwitchState	18	18	15	0.500	2.500	3.000

9.0 Observations and NESC Recommendations

9.1 Findings

The assessment team identified the following findings:

F-1. While there is often some degree of correlation, the number of SLOC does not indicate the complexity of the code. For example, straight-line code with zero decisions (e.g., no "if," "while," or "for" constructs) could possess a high SLOC count, but have a cyclomatic complexity equal to one.

F-2. There is a direct correlation between code complexity and the effort required to adequately test the code.

F-3. MC/DC testing is more suitable for safety-critical software when used in conjunction with functional and requirements testing, requires less time (i.e., 30–50%) to execute than BPT, is more robust than BPT, and reduces the redundant path testing of other code coverage criteria like MCC.

F-4. Where cyclomatic complexity is applied, it is typically used on safety- and mission-critical functions, with a maximum value between 10 and 20 typically chosen as the requirement or used in a coding standard.

F-5. Agency software verification focuses on meeting the requirements, but does not verify that every decision or branch of code performs correctly. Software developers are required to perform unit level testing, but no level of robustness or type of unit testing is specified.

F-6. Several NASA projects using MC/DC as a testing approach have found MC/DC to limit software coding errors by providing full path coverage.

F-7. Results from academic studies on code complexity metrics are inconclusive because of the difficulty of performing controlled studies across different application areas, programming team skill levels, and languages (e.g., C++, Ada, Java).

F-8. Low cyclomatic complexity values (i.e., less than 15) will simplify and expedite code verification and testing.

F-9. Code churn (i.e., how often code is changed) and inter-module dependencies are also effective indicators for tracking coding errors.

F-10. Prevailing industry standards (e.g., DO-178C) recommend, or even require, MC/DC for safety-critical software testing strategy.

F-11. Availability of tools to support MC/DC testing is more prevalent than for any other testing technique.

9.2 Observations

The assessment team identified the following observations:

O-1. The complexity of the architecture is also a critical element that should be considered. Dependency analysis can reveal that even cyclomatically simple code may be so interdependent as to be difficult to test or reuse. Hence, it is important to keep functions less structurally complex, but not if it forces the overall architectural complexity to become unmanageably interwoven.

O-2. Inadequate or unclear requirements are prone to cause software problems/errors later in the project life cycle.

O-3. In addition to the verification and risk reasons for limiting cyclomatic complexity, software maintainability efforts will indirectly benefit from lower complexity.

O-4. Software defects or anomalies discovered post-release reveals a flaw in testing and should be addressed either through additional testing, requirements, or both.

9.3 NESC Recommendations

The assessment team identified the following NESC recommendations, directed to NASA and NASA contractor software developers.

R-1. Use MC/DC testing methodology in conjunction with functional and requirements testing for software projects with safety-critical code. ***(F-2, F-3, F-6, F-10, F-11)***

R-2. Employ a cyclomatic complexity less than or equal to 15, or provide a credible rationale for not meeting that metric, for safety-critical software. Use in conjunction with other software metrics (see R-3) and recognize the possibility of changing the value based on future Agency software development. ***(F-2, F-4, F-8)***

R-3. Track and minimize code churn and inter-module architectural and design dependencies, keeping both to a minimum. ***(F-9)***

R-4. Run at least one, but preferably more, commercial static source code analyzers with a strict set of rules on every build and discuss the results in module code reviews; the ratio of warnings to SLOC should be minimized. ***(F-1, F-5, F-9, O-4)***

- Examples of suitable analyzers include: Coverity, Codesonar, Semmle, and KlockWork.

R-5. Consider using the Risk Management Framework found in NIST 800-37 when accounting for risk associated with cyclomatic complexity. ***(F-2, F-4, F-8)***

R-6. Focus requirement compliance on reducing risk rather than simply nontechnical features (e.g., checking for style and formatting). ***(F-1, O-1, O-2, O-3, O-4)***

R-7. Ensure that all code compiles without warnings, with warnings enabled at the highest possible level (e.g., gcc–Wall–pedantic) and map warnings to errors (i.e., compiler warnings stop the build). ***(O-1, O-2, O-3, O-4)***

R-8. Every defect or anomaly discovered post-release reveals a flaw in testing and should be addressed through additional testing, requirements, or both. ***(O-1, O-2, O-3, O-4)***

R-9. Complete test coverage for software safety-critical code should be required. ***(F-6)***

10.0 Recommended Updates to NASA Standards and Specifications

The assessment team recommends adding two new requirements to NPR7150.2, NASA Software Engineering Requirement, for current and future HEO programs/projects:

3.7.4 The project manager shall ensure that there is 100% code test coverage using the MC/DC criterion for all identified safety-critical software. (SWE-208)

Rationale: In MC/DC coverage, every condition in a decision must be tested independently to reach full coverage. Each condition must be executed twice, with the results true and false, but with no difference in the truth values of all other conditions in the decision. In addition, it must be shown that each condition independently affects the decision.

Aerospace and space guidance prioritizes safety in the SDLC. MC/DC represents a compromise that balances rigor and effort, positioning itself between DC and MCC. MC/DC requires a smaller number of test cases in comparison to MCC, while retaining a high error-detection probability.

3.7.5 The project manager shall ensure all identified safety-critical software components have a cyclomatic complexity value of 15 or lower for each software component. (SWE-209)

Rationale: Cyclomatic complexity is a metric used to measure the complexity of a software program. These metrics measure independent paths through the source code. The point of the requirement is to minimize risk, minimize testing, and increase reliability associated with safety-critical software code components, thus reducing the chance of software failure during a hazardous event. The software developer should assess all software safety-critical components with a cyclomatic complexity score over 15 for testability, maintainability, and code quality.

In addition, NPR 7150.2C, NASA Software Engineering Requirement, should be updated with detailed guideline information on architectural and cyclomatic complexity and MC/DC testing to guide developers in meeting these requirements and understanding their interactions.

11.0 Alternative Viewpoint(s)

There was one alternative viewpoint, as expressed in the following by Gerard Holzmann:

A minority of the team held that the most effective method for increasing code quality and reducing the residual defect density of software is not to bound the number of SLOC or the

cyclomatic complexity of functions, but rather to adopt strict adherence to the following development practices:

1. Ensuring compliance with a sensible coding standard for safety-critical code, focused specifically on risk reduction. The JPL Institutional Standard for the C Programming Language (JPL-D-60411, March 2009) is an example.
2. Running at least one, but preferably more, strong commercial static source code analyzers on every build of the code that include checkers for the rules in the coding standard used.
3. Requiring that all code be compiled with all available warnings in the compiler enabled at their highest level (e.g., -pedantic), while generating zero warnings.
4. Maintaining an average assertion density for all code modules of at least 2%. (Assertion density has been shown to correlate strongly with post-release fault density in studies done at Microsoft Research [ref. 4].)
5. Tracking all higher-level software requirements into the code and deriving test suites directly from the requirements. If full MC/DC code coverage is not realized in this way, it would mean requirements were incomplete or part of the code was redundant. In both cases, the issue should be analyzed and addressed. Note that a function that computes a square root and/or sorts data cannot be assumed to have been sufficiently tested if only full MC/DC coverage is realized: the function should be tested to actually compute the square root or sort the data, including in corner cases, to reject invalid inputs. This can be done only when tests are derived from higher-level requirements.

12.0 Other Deliverables

No unique hardware, software, or data packages, outside those contained in this report, were disseminated to other parties outside this assessment.

13.0 Definition of Terms

BPT	One form of structured, or "white-box" testing, driven by the code's cyclomatic complexity. The method analyzes the control flow graph of a program to find a set of linearly independent paths of execution. It has become a commonly used criterion for measuring software testing coverage.
Code Coverage	A measure used to describe the degree to which the source code is executed when a particular test suite runs. Software with high test coverage, measured as a percentage, has had more of its source code executed during testing, suggesting it has less chance of containing undetected software errors compared to a code with low test coverage.
Cyclomatic Complexity	Measures the number of decisions within the code based upon its control flow graph. Equal to the number of test cases required to test all linearly independent, or "basis," paths through the code.
Halstead Volume	Describes the size of the implementation of an algorithm. The computation is based on the number of operations performed and operands handled in the algorithm.

Linearly Independent	Each code path has at least one edge that is not in another path.
Software Testing	An investigation conducted to provide stakeholders with information about the quality of the software product or service under test.

14.0 Acronyms and Nomenclature List

APC	Autonomous Power Controller
BPT	Basis Path Testing
CC or CC1	Standard Cyclomatic Complexity
CC2	Strict Cyclomatic Complexity
CC3	Modified Cyclomatic Complexity
CFS	Core Flight Software
DC	Decision Coverage
GSFC	Goddard Space Flight Center
HEOMD	Human Exploration and Operations Mission Directorate
JPL	Jet Propulsion Laboratory
JSF	Joint Strike Fighter
LaRC	Langley Research Center
MC/DC	Modified Condition/Decision Coverage
MCC	Multiple Condition Coverage
MSFC	Marshall Space Flight Center
NESC	NASA Engineering and Safety Center
NPR	NASA Procedural Requirements
OFT	Orbital Flight Test
OSMA	Office of Safety and Mission Assurance
SC	Statement Coverage
SDLC	Software Development Life Cycle
SDLC	Software Development Life Cycle
SLOC	Source Lines of Code
UCC	Unified Code Counter

15.0 References

1. M. Shepperd and D.C. Ince, “A critique of three metrics,” J. Systems Software, 1994, Vol. 26, pp. 197-210.
2. M. Alfadel, A. Kobilica, and J. Hassine, “Evaluation of Halstead and cyclomatic complexity metrics in Measuring defect density,” 9th IEEE-GCC Conference, 2017.
3. Y. Tashtoush et al, “The correlation among software complexity metrics with case study,” Int. J. of Advanced Computer Research, Vol. 4, No. 2, June 2014.
4. G. Kudrjavets, N. Nagappan, and T. Ball, “Assessing the relationship between software assertions and code quality: an empirical investigation,” Proc. 2006 17th Int. Symp. On Software Reliability Engineering, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2006-54.pdf.
5. G. Holzmann, Test Fatigue, IEEE Software, 2020, Vol. 37, No. 4, pp. 11-16.
6. Personal communication, Dr. N. Nagappan (Microsoft Research), Ben Cichy (Blue Origin), Owen Cheng (Uber Advanced Technologies).
7. Better Embedded System SW, 2014, Phil Koopman (Carnegie Mellon), Jay Marchetti (Carnegie Mellon).
8. Identifying Error-Prone Software—An Empirical Study, IEEE Transactions on Software Engineering, Vol. SE-11, 1984, Vincent Y. Shen, Tse-Jie Yu, Stephen M. Thebaut, Lorri R. Paulsen.
9. Brader, Larry; Hilliker, Howie; Wills, Alan (March 2, 2013). “Chapter 2 Unit Testing: Testing the Inside.” Testing for Continuous Delivery with Visual Studio 2012. Microsoft. p. 30. ISBN 1621140180. Retrieved 16 June 2016.
10. National Institute of Standards and Technology (NIST) Special Publication 800-37, Guide for Applying the Risk Management Framework to Federal Information Systems.
11. A Complexity Measure, Thomas J. McCabe, IEEE Transactions on Software Engineering, Vol SE-2, No. 4, December, 1976.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
12/15/2020	Technical Memorandum	

4. TITLE AND SUBTITLE
Cyclomatic Complexity and Basis Path Testing Study

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Squire, Michael D.; Maynard-Nelson, Laura A.; Brown, Terry A.; Crumbley, Robert T.; Hozmann, Gerald J.; Jennings, Michael; Luu, Kequan; Moleski, Walter F.; Marchetti, Jay D.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
860921.01.23.01.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
NESC-RP-20-01515

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TM-20205011566

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61 Computer Programming and Software
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The NASA Chief Engineer requested the NASA Engineering and Safety Center (NESC) to conduct a study to determine the benefits of cyclomatic complexity and basis path testing (BPT) for software and whether they should be required. The principal focus of the assessment was to assess the use of cyclomatic complexity and BPT on safety-critical software. The purpose was to ensure that safety-critical software is not overly complicated to the point of increasing coding errors and that verification is more robust than for non-safety-critical software. This document contains the outcome of the assessment

15. SUBJECT TERMS
Cyclomatic Complexity; NASA Engineering and Safety Center; Basis Path Testing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	27	(443) 757-5802