

WHITEPAPER

Code-Based Test of CT

Document Information

Document

Distribution date July 18, 2025

Technical Support

Email help@suresofttech.com

Phone +82-31-606-2000

Legal Notice

© 2025. Suresofttech Inc. All rights reserved.

Proprietary and Confidential This material is proprietary to Suresofttech Inc. It contains trade secret and confidential information which is solely the property of Suresofttech Inc. This material is for client's internal use only. This material shall not be used, reproduced, copied, disclosed, and transmitted, in whole or in part, without the express consent of Suresofttech Inc.

Some images included in this document may differ slightly depending on the version. The image corresponding to the function being described is identical to the version specified in the document information.

Table of Contents

1	Overview	4
2	Code-Based Test of CT	6
	Basic Concept and Core Values	6
	Usage and Processing Flow	6
	Detailed Features and Characteristics	8
	Advanced Analysis Features	10
	Practical Impact	12
3	AI-Based Test Code Generation	13
	AI Chatbot-Based Test Code Generation	13
	Basic Usage	14
	Practical Impact	15
4	Summary of Core Values	17
	Improved Development Productivity	17
	Support for Standard Compliance	17
	Leveraging Existing Assets	18
	Improving Development Culture	18
	Cost Reduction	18
5	Conclusion and Future Plans	20
	Expected Value and Practical Impact of Adoption	20
	Future Directions	20

1 Overview

Testing in mission-critical domains is also evolving with the times. In industries such as automotive, aerospace, and medical devices, the software release cycle is rapidly shortening. In the automotive industry, the shift toward Software-Defined Vehicles (SDVs) and the adoption of OTA updates require continuous deployment. In the aerospace industry, companies like Lockheed Martin operate DevSecOps-based software factories to accelerate agile development cycles. In the medical device sector, the FDA has introduced a new regulatory framework to accelerate the lifecycle of AI-based devices.

C/C++ dominates software development in mission-critical domains. C is difficult to test due to low-level memory management and pointer operations. C++ adds further complexity through object-oriented features and templates, making it even harder for third parties to understand the code. These characteristics make testing C/C++ software particularly challenging.

As a result, it is difficult for testers who are not the original developers to test at the code level. The shortage of testing time caused by shortened release cycles further accelerates this situation. Naturally, the environment is shifting toward developers performing testing themselves.

Google Test (GTest), originally a widely used Code-Based Test framework in C++ development, is spreading in response to this shift. Many companies are accumulating GTest-based test assets and experience. It is familiar and flexible for developers, allows tests to be managed at the same level as the code, and enables rapid writing in IDEs or text editors. When introducing a new test tool, developers expect to continue working naturally in their familiar environment while leveraging existing GTest assets.

However, there are mandatory standards in mission-critical domains. Structural coverage requirements (such as MC/DC) and requirements traceability demanded by standards like ISO 26262 and DO-178C go beyond the scope of open-source test frameworks like GTest. Meeting these standards requires complex integration with external tools and verification of all tools used, resulting in significant additional cost and effort.

On the other hand, there are commercial test tools that meet the requirements of mission-critical domains. These tools provide traceability, coverage, and reporting features for standard compliance and have obtained tool qualification certifications. However, they come with inconveniences for developers. Most of them have proprietary test structures and require tests to be designed through GUI-based features. Even for simple tests that could be easily written in code, developers must learn how to use these tools and design tests accordingly. This forces them to work in an unfamiliar environment, reducing development productivity.

Developers are caught in a dilemma: either sacrifice productivity with inefficient tools or bear the cost of compliance using familiar tools.

Additionally, in the field of software testing, there is a growing trend of using AI to automatically generate test code. According to Stack Overflow's 2024 Developer Survey, 44% of developers are already using AI tools, and 75% plan to use them within the next 12 months. While AI is being applied in various areas, it is particularly strong at generating code. If test code can be generated and used directly via AI, testing productivity can increase significantly.

The Code-Based Test of CT resolves this dilemma. CT is an automated C/C++ unit and integration testing tool developed by Suresofttech. It is designed to support software testing and structural code coverage goals in mission-critical domains.

CT is certified by TÜV SÜD for the following functional safety standards: IEC 61508-3:2010 (Functional Safety – Software Requirements), ISO 26262-8:2018 (Automotive Functional Safety – Supporting Processes), IEC 62304:2015 (Medical Device Software – Lifecycle Processes), EN 50128:2011/A2:2020 (Railway Applications – Communication, Signaling and Processing Systems), IEC 60880:2006 (Nuclear Power Plants – Software for Systems Important to Safety), and IEC 62279:2015 (Railway – Control and Protection Systems). These certifications indicate that CT can be used to test the highest safety integrity requirements (e.g., SIL 4, ASIL D) across industries.

The Code-Based Test of CT, introduced in CT 2025.06, delivers three core values. First, developer-friendliness: developers can write GTest-style test code directly in a familiar format. Second, automated standard compliance: the written Code-Based Test is executed by CT's verified engine, automatically satisfying all standard requirements such as coverage and traceability. Third, productivity enhancement through AI-driven test code generation and fault analysis support.

The ultimate goal of the Code-Based Test of CT is to enable C/C++ developers to perceive testing not as a burdensome obligation but as a natural part of development, achieving both productivity and system safety.

2 Code-Based Test of CT

Basic Concept and Core Values

The Code-Based Test of CT resolves the dilemma presented in Chapter 1. It allows developers to write tests in the familiar GTest-style while naturally meeting the standard requirements of mission-critical domains.

The core idea is simple: developers write tests using common GTest macros such as TEST and EXPECT_EQ, just as they normally would. CT then converts this code to fit its own testing framework and executes it. Although the test code is written in GTest format, developers can fully leverage the powerful features of CT.

Table 1. Macro Mapping Between CT and GTest

CT MACRO	Equality assertion	EXPECT_LT
CS_EXPECT_EQ	EXPECT_EQ	Equality assertion
CS_EXPECT_NE	EXPECT_NE	Inequality assertion
CS_EXPECT_LT	EXPECT_LT	Less than comparison
CS_EXPECT_GT	EXPECT_GT	Greater than comparison

This approach addresses two pain points simultaneously. First, developers do not need to learn how to use a new tool—they can fully leverage their existing GTest experience. Second, standard requirements such as MC/DC and requirements traceability are naturally satisfied without the need for separate tool integration or qualification processes.

CT supports 44 out of the 94 total GTest macros, with full compatibility for the 17 most frequently used macros. This means that existing GTest users can write test code in CT without any additional learning.

Usage and Processing Flow

The usage of the Code-Based Test of CT is very simple.

Step 1: Write Test Code

```
TEST(MathTest, Addition) {  
    EXPECT_EQ(2 + 2, 4);  
    EXPECT_EQ(10 + 5, 15);  
}
```

The developer writes GTest-style test code using the CT editor or an external editor such as VS Code. It is managed just like a regular C++ source file.

Step 2: Automatic Parsing and Recognition

CT automatically parses the written test code and recognizes the file–suite–test case structure. No additional configuration or registration is required.

Step 3: Internal Conversion and Execution

When executing the test, CT converts the GTest macros into macros of its own testing framework. Users do not need to be concerned with this process, as the CT engine handles the execution.

Step 4: Result and Coverage Analysis

The execution results and coverage information are displayed in the CT UI. Advanced coverage metrics such as MC/DC are measured automatically. No additional work is required from the developer—just write and run tests as you normally would when using GTest.



Figure 1 Code-Based Test Execution Flow

Detailed Features and Characteristics

Macro Compatibility and Internal Conversion Structure

CT provides its own macros such as CTEST_SUITE, CTEST_CASE, and CS_EXPECT_EQ, but standard GTest macros can also be used without modification. Most macros familiar to developers—such as TEST, TEST_F, EXPECT_EQ, and ASSERT_TRUE—are supported.

While the tests appear to use GTest syntax, they are internally processed by CT's execution framework. This allows full utilization of CT-specific features such as coverage collection, integrated reporting, and execution environment control. This greatly reduces the training costs and resistance typically associated with adopting a new framework.

Code-Based Test View and Editor Integration

CT's Code-Based Test view provides a visual representation of tests in a file-suite-case structure, enabling fast navigation and easy management. Test cases are organized based on the actual source code structure, and clicking on a specific test case takes the user directly to its corresponding location in the editor.

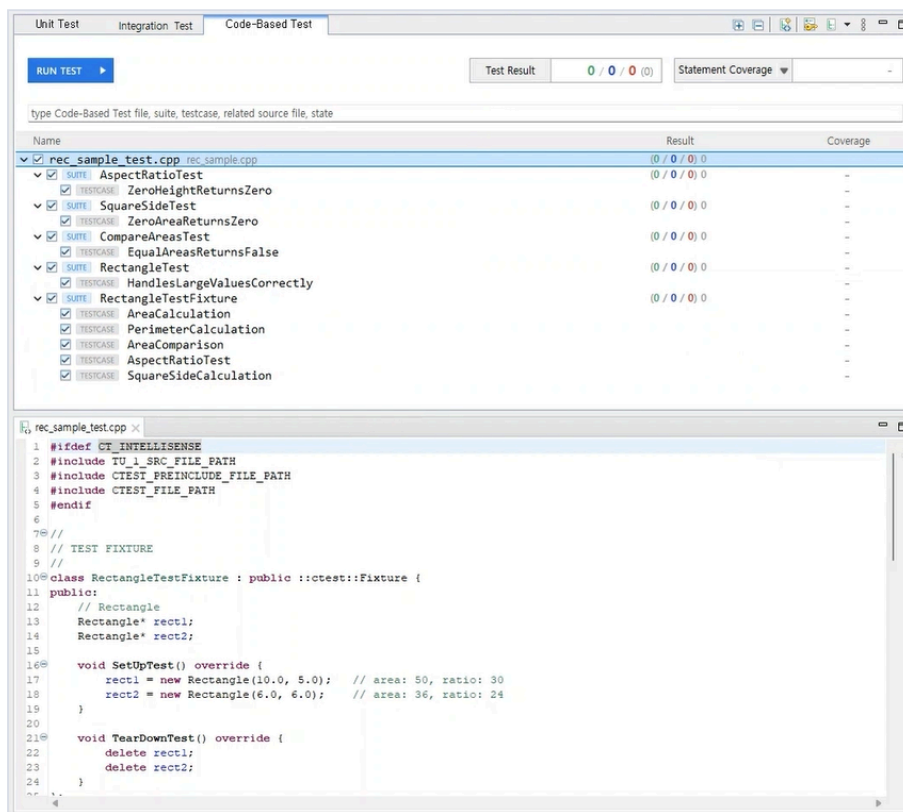


Figure 2 AI Code-Based Test View – Test Code Editing Screen

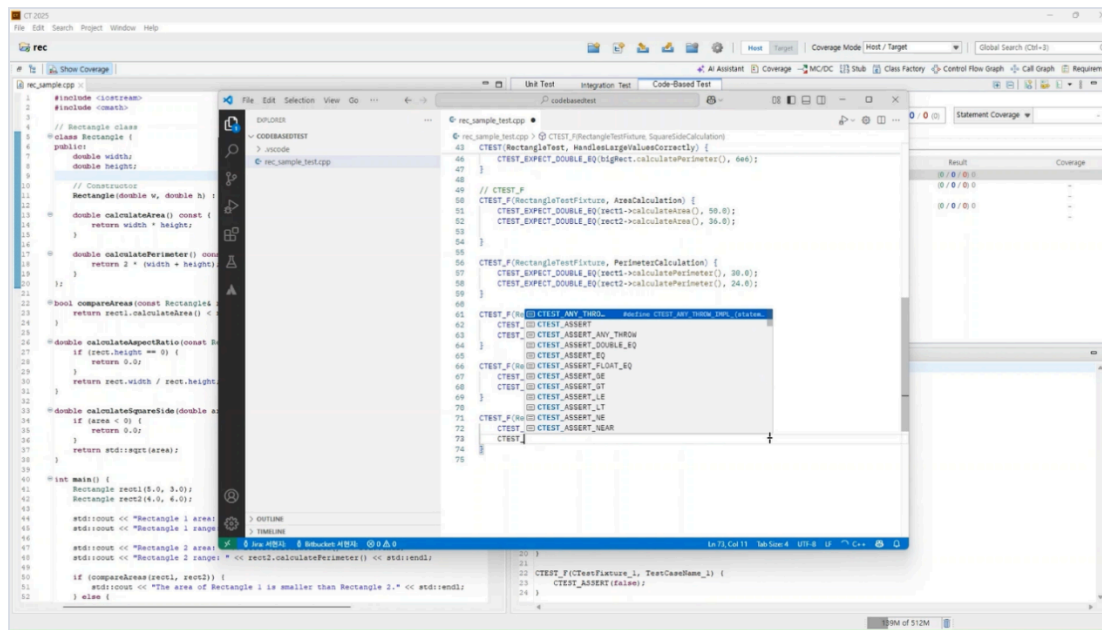


Figure 3 Open in VS Code Feature

Code-Based Tests can be opened and debugged directly in VS Code. Convenient features such as auto-completion are also available in VS Code. Since the test files are standard C++ code, they can be edited in any other editor as well.

Management and Portability of Test Assets

Code-Based Tests are managed as regular source code files. To use them in another project, you can either use CT's test import feature or simply copy and paste the code directly.

Advanced Analysis Features

Precise Coverage Measurement

Code-Based Tests fully support the same coverage measurement and result analysis features as traditional CT tests. GTest is simply a test framework, and additional tools must be manually integrated to perform coverage measurement. Moreover, it is difficult to find open-source coverage tools that support MC/DC. CT solves this problem by automatically measuring MC/DC coverage.

CT shows which parts of the code (branches and conditions) are covered by each test case.

By using this information, developers can efficiently add test cases to fill in uncovered areas and achieve their coverage goals.

Execution Result Visualization and Integrated Reporting

Test execution results are organized by suite and case. You can view everything in a single screen—whether each test passed or failed, if any errors occurred, how much coverage was achieved, and the results for each function. Through visualizations and reports, you can clearly understand the current testing status and obtain accurate information to meet your objectives. These reports are also useful for sharing verification results.

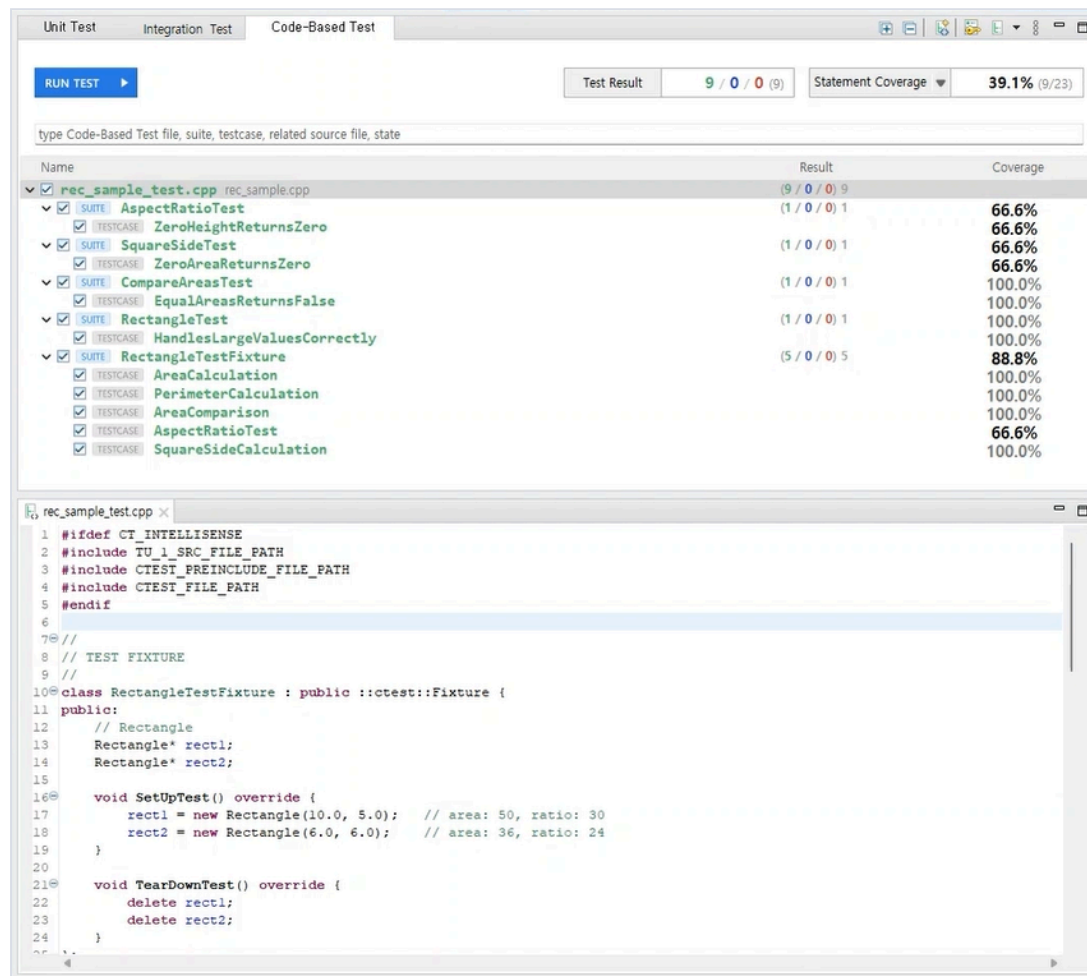


Figure 4 Code-Based Test Execution Results

Requirements Traceability Management

Code-Based Tests can be linked to requirements imported into CT to ensure traceability. Developers can meet standard traceability requirements without any additional manual effort.

Practical Impact

Improved Development Productivity

Existing GTest users face zero learning curve. They can write tests as usual while automatically benefiting from advanced features. Since tests can be written in familiar editor environments, the development workflow remains uninterrupted.

Renaming test cases, modifying inputs, or changing called functions is as straightforward as editing regular code. The barrier to maintenance is low, and tests can be rewritten or extended far more flexibly than with GUI-based test tools.

Automated Standard Compliance

The primary burden of standard compliance—mentioned in Chapter 1—is completely resolved. No external tool integration, separate coverage measurement, or tool qualification process is needed. All standard requirements can be fulfilled using CT alone.

Seamless Integration into Development Culture

Testing becomes a natural part of development rather than a separate task. Developers can write tests as they code and easily view the results. This lowers psychological barriers to testing and helps foster a quality-oriented development culture.

Leveraging Existing Experience

Developers familiar with the GTest style can apply their existing knowledge and experience without change. This minimizes learning costs and adaptation time when adopting a new tool.

The Code-Based Test of CT is a solution that achieves both developer friendliness and standard compliance. Without complex trade-offs or additional costs, developers can test in the way they prefer while naturally meeting mission-critical requirements. In doing so, CT contributes not only to improved code quality but also to the maturation of the development culture.

3 AI-Based Test Code Generation

AI Chatbot-Based Test Code Generation

AI technology is rapidly spreading throughout all areas of software development.

According to the 2024 Stack Overflow Developer Survey, 44% of developers are already using AI tools, and 75% plan to use them within the next 12 months.

AI is particularly valued for its practicality in repetitive and structured tasks such as writing test code.

CT reflects this trend by providing an AI chatbot-based test code generation feature.

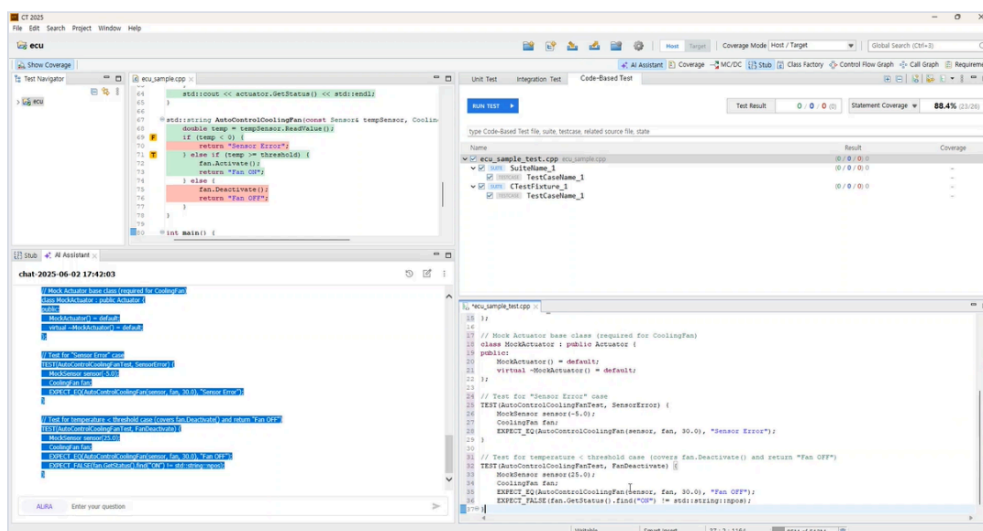


Figure 5 Coverage Results Before Running AI Assistant

This feature brings to life the possibility mentioned in Chapter 1—"If AI can be used to directly generate and utilize test code, testing productivity can be greatly improved."

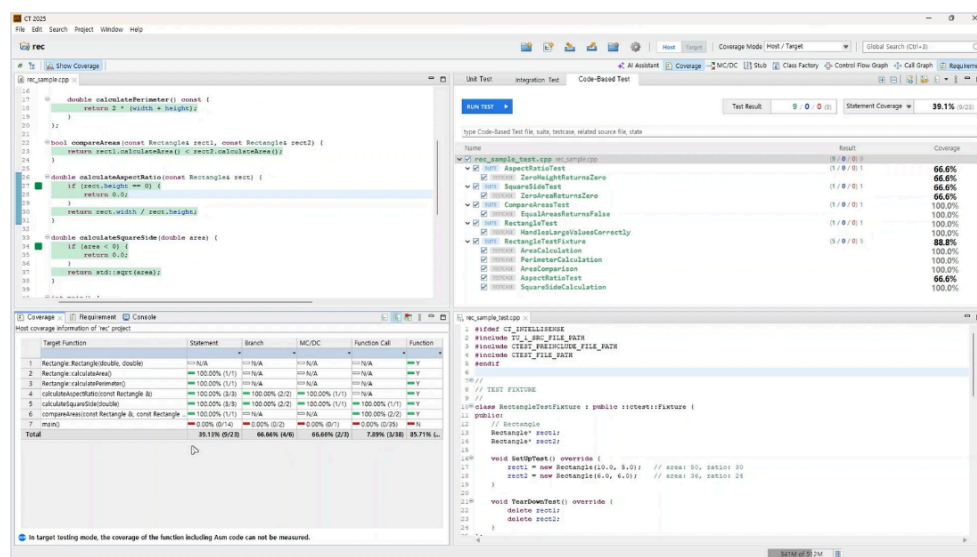


Figure 6 Coverage Results After Running AI Assistant

Basic Usage

For a target function, the "Generate Code-Based Test Code" feature allows AI to propose test code using CT's Code-Based Test macros (CTEST_SUITE, CTEST_CASE, CS_EXPECT_*).

The generated code is immediately executable and can be used as-is.

How AI-Based Test Code Generation Works

CT's AI feature does more than simply generate generic test code. It operates based on specific information aimed at achieving coverage goals.

Coverage Information Delivery Method

To perform precise coverage analysis, the AI receives structured data that includes relative line and actual code information. For example, the information is provided in the following format:

```
### 3. Uncovered Statements Information
- line 5:
    gz_state state;
- line 13:
    if (state->mode != GZ_WRITE || state->err != Z_OK)
- line 14:
    return 0;
```

Dependent Type Extraction and Management

CT performs recursive analysis to gather all type information necessary for test code generation. The data is optimized to include as much information as possible within the input token limits of the AI model.

Based on this structured information, the AI does not simply create generic tests, but instead generates meaningful test cases that can actually improve coverage.

Practical Impact

Reduced Burden of Repetitive Tasks

Manually analyzing uncovered code paths and writing corresponding test code is time-consuming. The AI feature automates this repetitive analysis process, allowing developers to focus on more critical tasks.

Maintaining a Code-Centric Testing Workflow

Test code generated by AI is also provided in the GTest-style format. Developers can continue working in a code-based environment without needing to switch to a separate GUI. Reviewing and modifying the generated code is done just like editing regular source code.

Increased Efficiency in Test Writing and Debugging

Users can ask natural language questions about test objectives or causes of test failures. The AI chatbot provides example code and suggestions, helping developers write and maintain tests more quickly and effectively.

Future Directions for Enhancing AI Capabilities

The currently available AI features are in an early stage and will gradually be expanded to provide more comprehensive support for test design.

Expanded Coverage Contribution of Suggested Test Code

Improvements are planned to enable automatically generated test code to meet a broader range of coverage criteria according to CT's measurement standards.

For example, automatic test code generation aimed at achieving MC/DC goals is under consideration.

Improved Intent Understanding and Scenario Diversity

Enhancements to the AI model are being reviewed to better reflect diverse user requests and suggest appropriate testing strategies even in complex code structures.

Evaluation of CT Integration and Automation

CT is assessing the technical feasibility and usability of allowing the proposed test code to be directly created or applied as a Code-Based Test within CT.

Test Generation from Requirements or Design Specifications

CT is also considering features to automatically generate test code from requirements or design specification documents. This includes generating tests based on functional safety design methods or producing additional test code to improve coverage.

In such cases, traceability to requirements can also be automatically established, further reducing the developer's workload.

These directions aim to harness the potential of AI while maintaining CT's core values of reliability and precision. Improvements will be made progressively, with full consideration of user needs and tool suitability.

CT's AI-Based Test support reduces the time and effort developers spend on writing tests, while helping them create higher-quality tests. By implementing the vision of "improving test productivity with AI" presented in Chapter 1 as a tangible feature, CT enhances the value of Code-Based Testing.

4 Summary of Core Values

Improved Development Productivity

Significant Reduction in Learning Effort

CT's Code-Based Test can be used immediately by existing GTest users without additional learning. Developers can write tests using familiar macros such as TEST and EXPECT_EQ, while also taking advantage of CT's advanced features. This greatly reduces the "learning burden of new tools" discussed in Chapter 1.

Maintaining Development Workflow

Developers can write tests just like writing code in their familiar editor environment. There is no need to switch to a GUI tool, allowing for a natural and continuous workflow. Test maintenance is done just like editing regular code, requiring no additional learning or complicated procedures.

Increased Efficiency Through AI

AI features support repetitive test-writing tasks. With AI assisting in coverage analysis and test case design, developers can focus on more critical tasks such as architecture and review.

Support for Standard Compliance

Eliminating Tool Integration Complexity

Using GTest typically involves external coverage tool integration, separate traceability systems, and tool qualification processes. CT reduces these burdens by providing built-in features that help satisfy standard requirements.

Support for Advanced Coverage Measurement

CT automatically measures MC/DC, which is difficult to achieve in standard GTest environments. Advanced coverage analysis, which is hard to support with open-source tools, is provided with no additional configuration. Detailed per-test-case coverage contribution is also available, giving developers the insight needed to achieve their goals.

Requirements Traceability Management

CT's requirements traceability feature also applies to Code-Based Tests. Developers can meet the traceability requirements defined by standards without any additional manual effort.

Leveraging Existing Assets

Utilizing GTest Experience

Developers familiar with the GTest style can fully apply their existing knowledge and experience. Previously acquired skills such as test-writing techniques, debugging methods, and code structuring practices can be reused without change.

Flexible Test Asset Management

Code-Based Tests are managed as regular source code files, making them easy to move, back up, or share across projects. They can be systematically managed through CT's import/export features, or reused directly by copying and pasting the code when needed.

Improving Development Culture

Lowering the Barrier to Writing Tests

Testing becomes a natural part of development rather than a complex separate task. Developers can write tests just as they code, without the burden of learning new tools or environments, which reduces psychological resistance to testing.

Fostering a Quality-Driven Development Environment

As writing tests becomes easier, developers tend to write tests more frequently and more thoroughly. AI-generated suggestions help uncover overlooked test cases, enhancing awareness of code quality and promoting a quality-first mindset in development.

Cost Reduction

Reduced Tool Adoption Costs

CT provides both a developer-friendly testing environment and support for standard compliance within a single tool, reducing the need to purchase or integrate additional tools. This helps save on license fees, integration development costs, and ongoing maintenance costs typically associated with combining multiple tools.

Lower Training and Transition Costs

Existing GTest users can use CT without any additional training, minimizing time and expenses for education. Gradual adoption is also possible, so there is no need to halt ongoing projects or undertake large-scale transitions.

Efficient Maintenance

Since maintenance can be done in a familiar development style, the long-term management burden is reduced. Tests can be modified and extended with the same ease as editing regular code.

The Code-Based Test of CT provides a practical solution to the long-standing conflict between developer convenience and standard compliance in mission-critical domains. It enables developers to test in the way they prefer while still meeting rigorous industry standards—ensuring both productivity and quality.

5 Conclusion and Future Plans

Expected Value and Practical Impact of Adoption

The Code-Based Test feature of CT addresses a fundamental dilemma in mission-critical domains. Developers can write tests in the familiar GTest style while still being fully supported by the strictest standard requirements.

Balancing Developer Friendliness and Standard Compliance

The dilemma introduced in Chapter 1—having to choose between sacrificing productivity with inefficient tools or bearing the cost of compliance with familiar ones—has now been resolved. Developers no longer need to choose one over the other.

Differentiated Value Through AI Integration

AI-generated test code improves productivity. By analyzing coverage information and suggesting meaningful test cases, the AI feature reduces repetitive analysis and allows developers to focus on more critical tasks.

Practical Benefits of Adoption

Existing GTest users can adopt CT with virtually no learning curve, significantly lowering the adoption barrier. Since standard requirements can be fulfilled without complex tool integrations, both time and cost are saved.

Future Directions

Utilizing Legacy GTest Test Code

We are currently exploring ways to import and reuse existing GTest test code within the CT environment. Technical considerations such as differences in build environments and macro compatibility are being taken into account as we evaluate phased integration support strategies.

Plans for Advancing AI Capabilities

AI features will be expanded to include automatic test code generation aimed at achieving MC/DC coverage, and test generation based on requirements documents. By enabling automatic requirements traceability linking, CT aims to further reduce the burden on developers.

Continuous Improvement of Developer Experience

CT will continue to enhance developer-friendliness by expanding support for VS Code integration, reviewing compatibility with other major IDEs, and increasing compatibility with additional GTest macros.

The goal of CT's Code-Based Test is to make testing a natural part of the development process in mission-critical software development. By achieving both development productivity and system safety, CT contributes to building safer and more reliable software.