

WHITEPAPER

AI-Powered Software Testing: CT's AI Assistant Feature

Document Information

Document

Distribution date June 24, 2024

Technical Support

Email help@suresofttech.com

Phone +82-31-606-2000

Legal Notice

© 2024. 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.

Reproduction and distribution without permission from Suresofttech Inc. is prohibited.

For suggestions or inquiries regarding this document or product, please contact 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	Difficulties in C/C++ Testing	6
	Complexity of C/C++ Code	6
	Complexity of Embedded Environments	7
3	Role and Benefits of AI	9
	Reducing Test Environment Setup Time	9
	Improving Test Coverage	9
	Supporting Error Analysis and Resolution	9
	Requirements-Based Test Automation	10
4	Limitations and Challenges of AI Application	11
	Data Quality and Quantity Issues	11
	Integration Issues with Complex Systems	11
	Limitations in Predictability and Explainability	11
	Cost and Time Issues	12
5	CT's AI Assistant	13
	Introduction to CT	13
	Introduction to AI Assistant	14
6	Conclusion	20

1 Overview

The rapid evolution of software development methods has made it increasingly challenging to achieve both quality and efficiency simultaneously. Software complexity is increasing, release cycles are shortening, and user expectations are rising. Consequently, traditional software testing methods are struggling to perform sufficient testing to meet quality goals. Software testing faces many challenges due to high maintenance costs and frequent code changes.

AI technology is gaining attention as one solution to these problems. With the rapid advancement of AI technology in recent years, it's bringing significant changes to the field of software testing. AI can greatly improve the efficiency of software testing by automating repetitive tasks, analyzing large amounts of data to derive meaningful results, and quickly adapting to changing environments.

Table 1. Evolution of Software Testing Methods

Era	Testing Method
1990s	Manual Testing - People directly write and execute test cases
2000s	Automated Testing Tools - Test automation using scripts
2010s	CI/CD and DevOps - Rapid test cycles with continuous integration and deployment
2020s	AI-based Testing - Automation and predictive analysis using AI, intelligent configuration of test environments

The application of AI in software testing is expected to solve many existing problems and significantly reduce testing time. For example, AI can automatically generate code to replace complex dependencies and tests to increase test coverage. It can also suggest solutions for errors that occur during the testing process or automatically modify tests according to code changes to maximize efficiency.

With the introduction of AI, software development teams can reduce test maintenance costs, shorten test cycles, and ultimately deliver higher quality software to customers faster. AI will play an important role not just in verifying code, but throughout the entire development process.

This whitepaper explores the role and importance of AI technology in software testing. In particular, it explains the difficulties of testing C/C++ code, which is widely used in mission-critical fields, and suggests ways to utilize AI for this problem. We will introduce the AI assistant feature implemented in CT, a unit and integration test automation tool, and look at specific examples of how AI is being applied in actual products. The purpose of this whitepaper is to present ways to improve the efficiency and quality of software testing through AI technology.

2 Difficulties in C/C++ Testing

C/C++ languages are widely used in mission-critical fields due to their excellent performance and ability to directly control hardware. However, these advantages of the language can become disadvantages in the software testing process. In this chapter, we will take a detailed look at the difficulties of C/C++ software testing and examine how these difficulties affect the overall quality of the software.

Complexity of C/C++ Code

While C/C++ languages are powerful and flexible, their complexity poses challenges in testing. The main challenges related to the complexity of C++ code are as follows:

Memory Management

In C/C++ languages, programmers directly manage memory. This increases the likelihood of problems such as memory leaks or incorrect memory access. These issues are particularly difficult to detect and debug as they often only appear under specific conditions. For example, memory leaks caused by not freeing dynamically allocated memory usually don't manifest immediately but have an increasing impact over time, making it difficult to trace the source of the problem.

Pointer Arithmetic

Pointers are one of the powerful features of C/C++. However, incorrect pointer arithmetic can lead to unpredictable program behavior. For instance, if a pointer references an incorrect memory address, the program may terminate immediately or data may be corrupted. Moreover, complex pointer arithmetic makes it very difficult to track the execution path of the code, making it challenging to identify the cause when problems occur.

Low-Level System Access

C/C++ provides functionality to work closely with hardware. While this allows for high performance, debugging errors that occur in the interaction between hardware and software is complex. Additionally, incorrect hardware access can have a critical impact on the entire system.

Dependency Management

In large C/C++ projects, complex dependencies form between modules. Changes in one module can affect several other modules, making it difficult to write and maintain test cases. These complex dependencies increase the likelihood that code modifications will affect unexpected areas, increasing the risk of errors.

Legacy Code

Old code written in C/C++ often lacks documentation or has unclear code intentions. Maintaining and testing such code comes with considerable difficulties. Insufficient documentation requires a lot of time to understand the exact behavior of the code, which can degrade the quality of testing.

Complexity of Embedded Environments

C/C++ is particularly widely used in embedded systems. In embedded environments, software must be optimized for specific hardware, which makes the testing process even more complex. The following are the main challenges of testing C/C++ code in embedded environments:

Various Compilers and Target Boards

Embedded systems run on various hardware platforms, each with its own unique optimization options and characteristics. This diversity can cause the same source code to behave differently in different environments. For example, code that works normally with one compiler may cause errors with another compiler, increasing the complexity of the test environment.

Limited Resources

Embedded systems typically use limited CPU, memory, and power resources. These constraints require efficient operation of the software, and to meet this, the code must be highly optimized. Such optimization can reduce the readability and testability of the code. Also, due to limited resources, the use of debugging tools or additional logging for testing is restricted, increasing the difficulty of testing.

Real-time Performance Requirements

Many embedded systems must meet real-time performance requirements. This means tasks must be completed within a specific time, and if these requirements are not met, the system may not function normally. Real-time performance testing is a very sophisticated and complex task, requiring a precise test environment to verify accurate timing and behavior.

Identifying Causes of Errors Between Tests

In embedded systems, even the same test can yield different results depending on the environment and conditions. The main causes of this phenomenon are as follows:

1. **Compiler Optimization:** Using certain optimization settings, the compiler can modify the code to improve performance. In this process, the execution order or memory usage patterns of the code can be changed.
2. **Memory Access Method:** The order or method of memory access can be changed during optimization. This can cause data race conditions, especially in multithreading environments.
3. **Hardware-Dependent Code:** Code optimized for specific hardware may behave differently in other environments. For example, code tailored to a specific processor architecture may be interpreted differently on another architecture.

These factors each affect the behavior of the code in different ways. Compiler optimization can directly change the execution path, changes in memory access methods can affect data consistency, and hardware-dependent code can produce different results depending on the environment.

As a result, even the same source code may behave differently than expected in different environments, which can sometimes manifest as errors.

It's difficult to identify the cause of these problems because it's hard to reproduce the exact conditions under which errors occur. It requires understanding and analyzing the complex interactions between hardware and software, which is a time-consuming and resource-intensive task.

Testing C/C++ code poses many difficulties due to its complexity and various usage environments. Especially in embedded environments, identifying the cause of errors that occur during testing becomes even more challenging. Therefore, high levels of skill and experience are required to perform effective testing. Introducing AI technology to overcome these difficulties can greatly help improve the testing process and enhance software quality.

3 Role and Benefits of AI

The introduction of AI in software testing is bringing revolutionary changes to the testing process. By utilizing AI technology, we can automate repetitive tasks, derive meaningful insights from vast amounts of data, and adapt to dynamically changing environments. This can greatly improve the efficiency and accuracy of the testing process. In this chapter, we will discuss in detail the benefits and effects that can be gained by utilizing AI in software testing.

Reducing Test Environment Setup Time

AI technology can be used to automatically generate code that replaces the dependencies of the system under test (SUT). This allows for quick and reliable configuration of the test environment. With AI, you can efficiently set up the test environment without deep understanding of legacy code, significantly reducing the time required for environment setup. For example, by using AI to automatically generate mock objects and efficiently manage dependencies, you can more easily configure the test environment. Additionally, AI can detect and automatically correct dependency issues that may arise when code changes, maintaining a consistently stable test environment.

Improving Test Coverage

AI technology can be used to automatically generate new test cases to increase test coverage. By using AI to analyze the code base, identify untested parts, and then generate new test cases based on this, coverage can be increased. Furthermore, by analyzing existing tests, unnecessary test cases can be removed or tests that need improvement can be identified, thereby increasing test efficiency and maintaining optimal test coverage. AI tools can dynamically update test cases according to code changes and keep them up to date.

Supporting Error Analysis and Resolution

AI technology can be a powerful tool for analyzing and resolving errors that occur during the testing process. Here are examples of how AI can be used to analyze errors and obtain solutions:

1. **Log Analysis and Pattern Recognition:** AI can analyze large volumes of test logs to identify repetitive error patterns, group similar errors, and prioritize them, saving developers' time.
2. **Error Cause Tracing and Solution Suggestion:** AI can analyze error logs, code structure, and previous resolution cases to identify possible causes and provide developers with specific solution suggestions or code modification proposals.

3. Performance Bottleneck Analysis: AI can analyze performance data during integration testing to identify bottleneck points, identify code areas that need optimization, and suggest improvement measures.

Such rapid error identification and response can greatly contribute to reducing testing time.

Requirements-Based Test Automation

AI technology can be used to analyze requirements and specifications written in natural language and automatically generate test cases. Through AI, test cases can be generated based on requirements, and tests can verify whether the requirements are properly reflected in the code. For example, by using Natural Language Processing (NLP) technology to analyze specifications and automatically write test cases accordingly, it can verify whether requirements are accurately implemented. This is particularly useful when testing complex scenarios. Additionally, when code changes, existing tests can be used to verify whether the code has been correctly modified, allowing quick identification and correction of parts where requirements have not been properly reflected.

The introduction of AI can make the software testing process more efficient and rapid. However, despite the many benefits that AI technology offers, there are several challenges and considerations when actually implementing it. In the next chapter, we will look at the main limitations and difficulties that may be encountered when applying AI to software testing, and discuss how these issues can be managed.

4 Limitations and Challenges of AI Application

While AI provides many benefits in software testing, there are some limitations and difficulties to consider when implementing it. Understanding and managing these factors is crucial for successful AI application.

Data Quality and Quantity Issues

The performance of AI systems heavily depends on the quality and quantity of training data. Applying AI to software testing requires sufficient and diverse test data. However, when applying it to actual projects, you may face the following problems:

1. **Lack of Data:** It can be difficult to secure sufficient test data for projects in the initial stages or new systems.
2. **Data Bias:** If the AI model is excessively trained on certain types of data, its predictions may be biased or only work well in specific situations.
3. **Data Quality:** Incomplete or inaccurate data can negatively affect the learning and performance of AI models.

Integration Issues with Complex Systems

When integrating AI into existing complex software systems, the following difficulties may arise:

1. **Understanding Interactions:** It's necessary to clearly understand how the AI system will interact with the existing test environment.
2. **Interoperability Issues:** Effective methods for connecting AI tools with existing test systems must be devised.
3. **Increased Complexity:** The addition of AI tools can increase the complexity of the entire system, which can add additional burden to maintenance and management.

Limitations in Predictability and Explainability

The lack of transparency in AI models' decision-making processes can cause the following problems:

1. **Black Box Problem:** The internal workings of AI are not clear, which can reduce the reliability of results.

2. **Difficulty in Explanation:** It can be difficult to clearly explain AI's recommendations or decisions, making it challenging to accept the results.
3. **Accountability:** For important business decisions, it should be possible to clearly understand and explain the basis for AI's judgments.

Cost and Time Issues

Introducing AI technology may require significant initial investment:

1. **Initial Costs:** Large costs may be incurred for infrastructure construction, training data preparation, tool purchases, etc.
2. **Time Investment:** It may take considerable time to effectively implement and integrate AI systems.
3. **Delayed ROI Realization:** It may take time to realize return on investment, and productivity may decrease in the short term.

AI technology is a powerful tool that can bring revolutionary changes to software testing. However, various challenges such as data quality, complexity of system integration, limitations of predictability, and initial investment costs must be considered. By effectively managing these challenges, the effects of AI adoption can be maximized.

In the next chapter, we will explore how CT's AI assistant feature overcomes these limitations and difficulties, and how AI has been successfully applied to software test automation tools.

Introduction to AI Assistant

CT's AI assistant feature is an innovative function designed to efficiently solve various problems that users may face in the test process, improve test quality, and increase development productivity. When problems occur during the testing process, users can receive guidance for resolution through the AI assistant feature.

For this, CT performs the following process:

1. Analyze error messages
2. Collect error-related information
3. Generate prompts to be sent to the AI language model
4. Send system prompts and user prompts to the AI language model
5. Provide AI's response to users to support error cause identification and correction

The main benefits of the AI assistant are:

1. Reduced problem-solving time
2. Decreased need for external support requests
3. Enhanced user ability to utilize CT
4. Continuous improvement in test process efficiency

AI Assistant Strategy

The core strategy of the AI assistant is to automatically generate optimized prompts by selecting appropriate information and code snippets according to the situation. CT's AI assistant feature uses the following strategies to provide accurate and useful guidance:

1. Context Awareness: Recognize the context of the problem situation the user is facing and reflect this in the prompt.
2. Customized Information Collection: Analyze the problem situation and automatically generate prompts that include highly relevant information and code snippets.
3. Domain-Specific Prompts: Apply system prompts specialized for the C/C++ code testing domain to induce responses with high accuracy and relevance.
4. Continuous Improvement: Continuously improve the prompt generation algorithm through user feedback collection and testing.
5. AI Model Learning: Continuously train the AI language model based on new data and feedback, providing more sophisticated solutions over time.

AI Assistant Example

Here's an example of modifying a build stub that needs correction according to AI guidance, then re-running the test to verify that the build stub has been correctly modified.

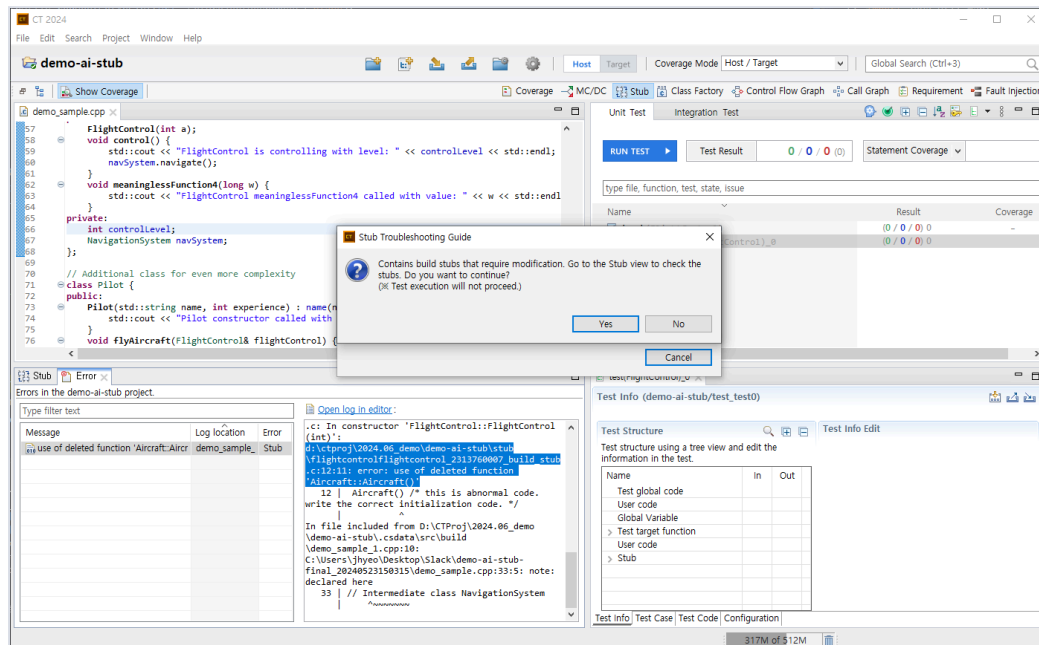


Figure 2 Identifying a build stub that needs modification

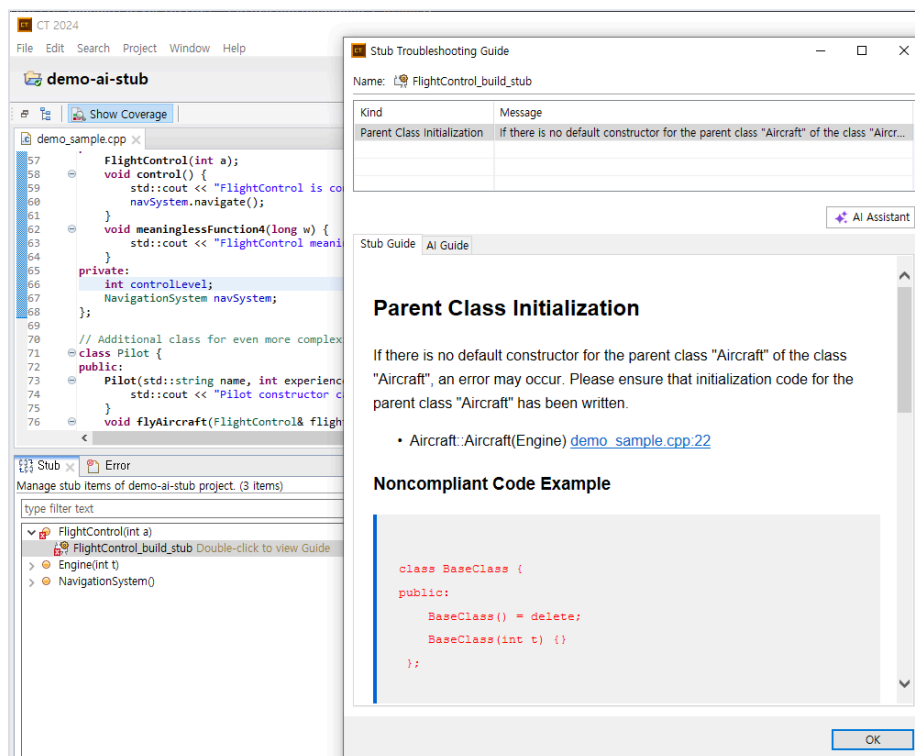


Figure 3 Executing the AI assistant in the stub problem-solving guide window

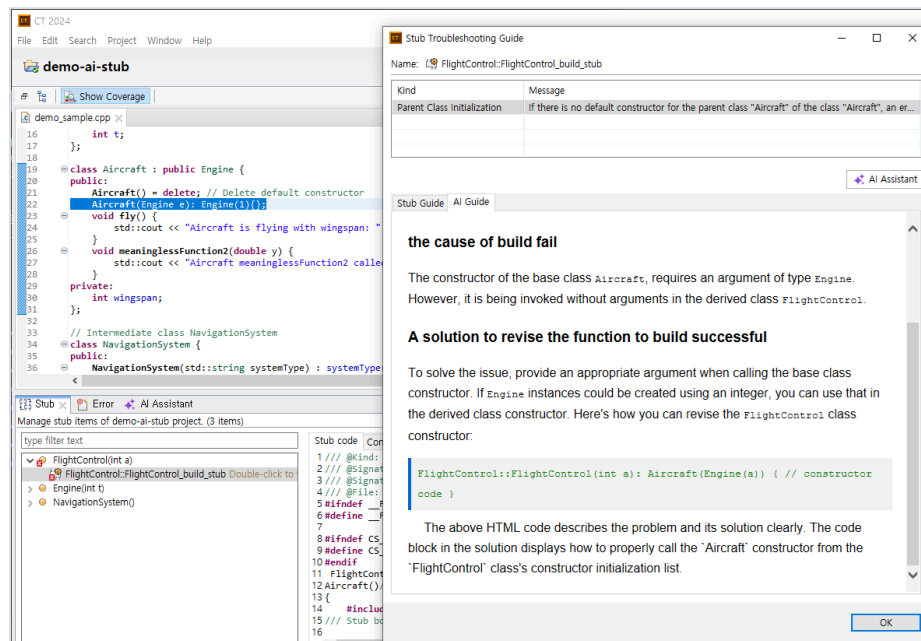


Figure 4 Checking the AI guide

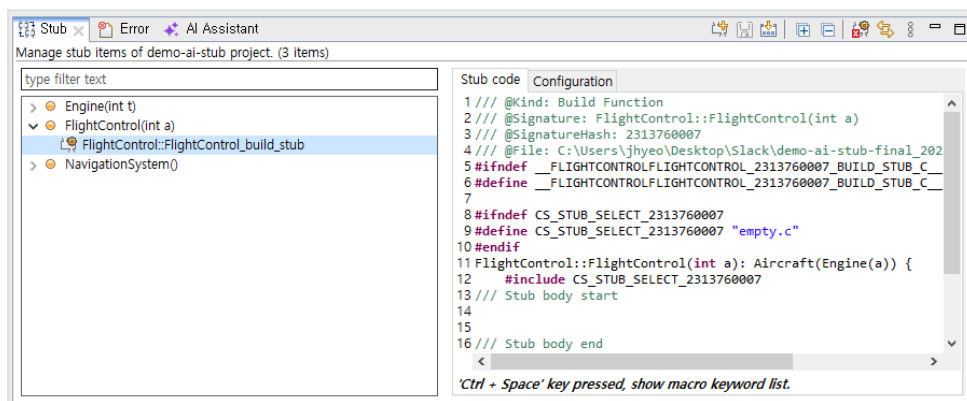


Figure 5 Modifying the build stub code referring to the AI guide

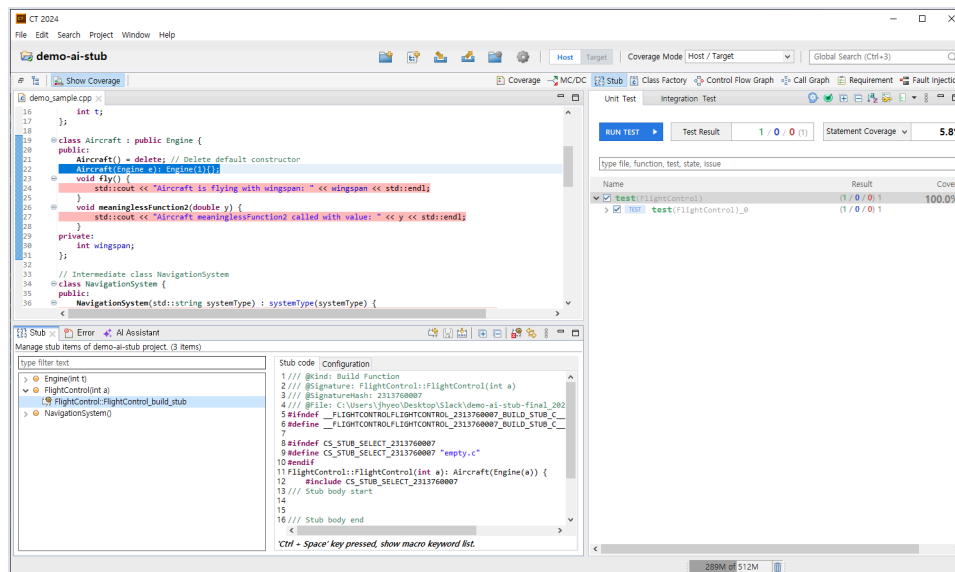


Figure 6 Test execution completed without build stub issues

Here's an example of using the AI assistant to resolve an error for a failed test case:

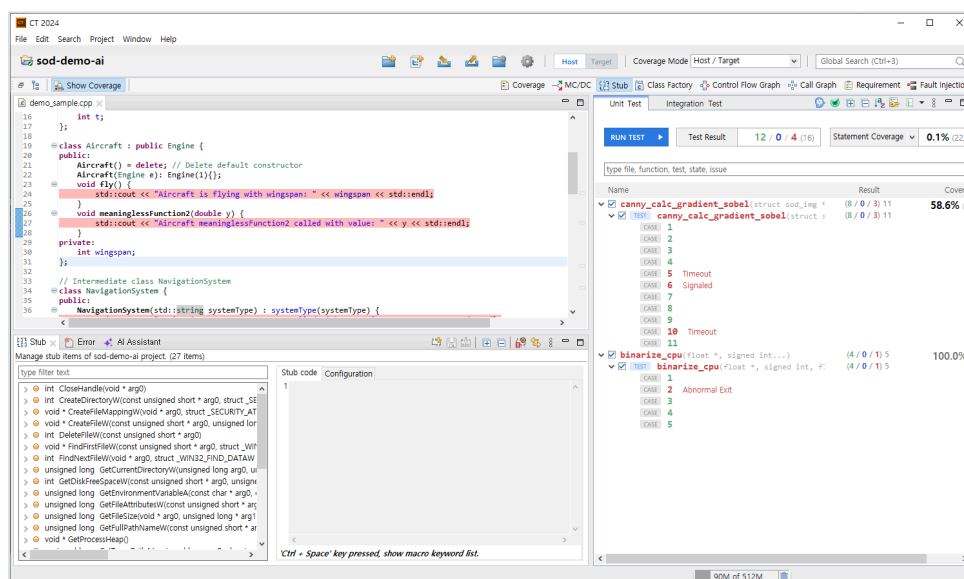


Figure 7 Identifying the test case that failed during test execution

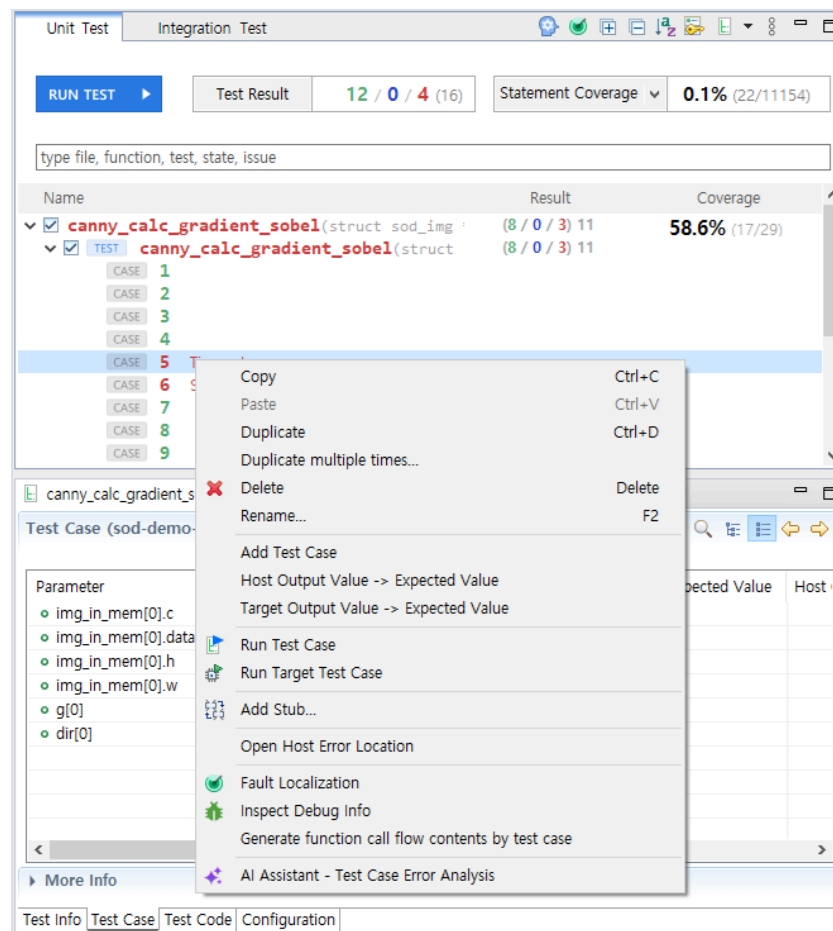


Figure 8 Executing the AI assistant for the failed test case

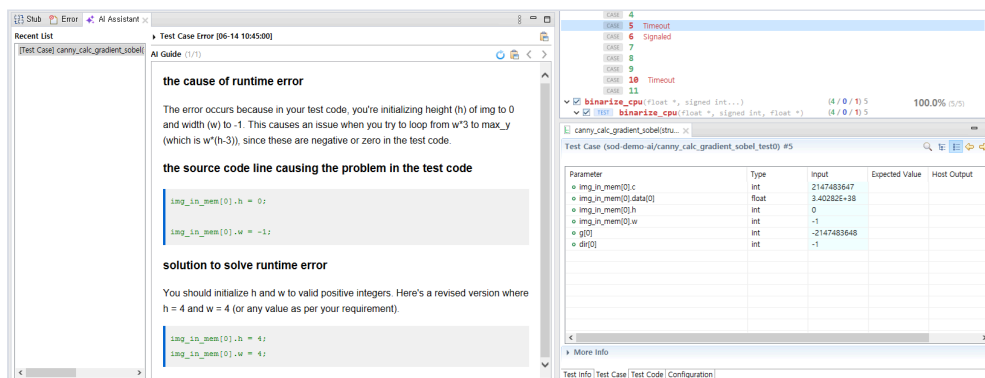


Figure 9 Modifying the test case referring to the guide

The screenshot displays the Suresoft testing interface. At the top, there are tabs for 'Unit Test' and 'Integration Test'. Below these, a 'RUN TEST' button is visible. The 'Test Result' section shows '13 / 0 / 3 (16)' and 'Statement Coverage' is '0.1% (22/11154)'. A search bar is present with the placeholder text 'type file, function, test, state, issue'.

The main test results table is as follows:

Name	Result	Coverage
✓ canny_calc_gradient_sobel (struct sod_img *, sign	(9 / 0 / 2) 11	58.6% (17/29)
✓ TEST canny_calc_gradient_sobel (struct sod_img	(9 / 0 / 2) 11	
CASE 1		
CASE 2		
CASE 3		
CASE 4		
CASE 5		
CASE 6	Signaled	
CASE 7		
CASE 8		
CASE 9		
CASE 10	Timeout	
CASE 11		
✓ binarize_cpu (float *, signed int...)	(4 / 0 / 1) 5	100.0% (5/5)
✓ TEST binarize_cpu (float *, signed int, float *)	(4 / 0 / 1) 5	

Below the test results, a detailed view for the test case 'canny_calc_gradient_sobel(stru...' is shown. It includes a table with parameters, types, inputs, expected values, and host outputs.

Parameter	Type	Input	Expected Value	Host Output
img_in_mem[0].c	int	2147483647		
img_in_mem[0].data[0]	float	3.40282E+38		
img_in_mem[0].h	int	4		
img_in_mem[0].w	int	4		
g[0]	int	-2147483648		
dir[0]	int	-1		

At the bottom, there are tabs for 'Test Info', 'Test Case', 'Test Code', and 'Configuration'.

Figure 10 Test case successful without errors

6 Conclusion

In this whitepaper, we have deeply explored the role and importance of AI technology in software testing. In particular, we have detailed the complexity of C/C++ code testing and CT's AI assistant feature, an innovative solution to overcome this.

The modern software development environment is becoming increasingly complex, and release cycles are getting shorter. In these changes, it has become difficult to ensure sufficient quality assurance with traditional testing methods alone. AI technology is gaining attention to respond to these challenges, and its potential is enormous. AI can dramatically improve the efficiency of the testing process through its ability to automate repetitive tasks, quickly analyze large amounts of data, and flexibly adapt to changing environments.

The introduction of AI technology can solve many difficulties faced by developers and testers by effectively managing complex dependencies, greatly improving test coverage, and quickly supporting error analysis and resolution. However, there are still some important challenges in adopting AI. These include securing data quality, complex integration processes with existing systems, limitations in the predictability and explainability of AI models, and costs associated with initial adoption. Effectively managing these limitations and difficulties is a crucial factor in the success of AI adoption.

CT effectively integrates this AI technology to manage the complexity of C/C++ code testing. CT's AI assistant feature significantly reduces test environment setup time, improves test coverage, and effectively supports error analysis. In particular, it provides real-time customized guidance for problem-solving and continually improves its performance through ongoing learning. Through this, CT supports development teams in delivering higher quality software faster.

The combination of AI and CT is expected to lead the future of software testing. This will play an important role in organizations meeting increasingly complex and demanding software development requirements. Through Suresofttech's CT and its AI assistant feature, development teams can maximize the efficiency and quality of their test processes and enhance their ability to rapidly deliver high-quality software.

In conclusion, the advancement of AI technology and CT's innovation are fundamentally changing the paradigm of software testing. This will contribute to setting new standards across software development, beyond simply improving the testing process. We expect that the software development and testing fields will continue to evolve through the synergy effect of AI and CT.