WHITEPAPER

Testing Automotive AI Software for ISO/PAS 8800

Building Traceable Software Verification Evidence with CT

© 2026. Suresofttech Inc. All rights reserved.

Document Information

Document

Distribution date	November 9, 2026

Technical Support

Email	help@suresofttech.com
Phone	+82-31-606-2000

Legal Notice

© 2026. Suresofttech Inc. All rights reserved.

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

For suggestions or inquiries regarding this document or product, please contact Suresofttech Inc.

This whitepaper provides a technical explanation of ISO/PAS 8800:2024 and describes how to obtain software verification evidence using CT. The content of this whitepaper alone does not guarantee ISO/PAS 8800 conformity, product certification, or vehicle-level safety.

The features supported by CT may vary depending on the product version, license, target environment, and integration configuration. Actual projects should base their decisions on the original standard, the project safety plan, and the applied CT version and its certification scope.

Summary

ISO/PAS 8800 addresses the safety of automotive AI by considering data, models, software, and operating conditions together. Model performance evaluation and verification of the surrounding software have different verification targets, and their results must be linked to safety requirements and must come from the same configuration of the item under verification.

CT (Controller Tester) is a C/C++ testing tool certified by TÜV SÜD in accordance with ISO 26262. CT links the requirements of the C/C++ software surrounding an AI model to tests, and produces execution results and structural coverage for normal and abnormal conditions. This whitepaper first explains the verification and evidence required from the ISO/PAS 8800 perspective, and then shows how to obtain software verification evidence with CT and use it in a project's assurance argument.

© 2026. Suresofttech Inc. All rights reserved.

Table of Contents

1 Why Software Verification Matters for Automotive AI Safety 5

2 ISO/PAS 8800: Explaining AI Safety with Evidence 7

Scope and Relationship to Existing Standards 7

Required Verification and Safety Evidence 7

3 Software Verification Evidence Obtained with CT 11

Preparing the Verification Environment 11

Running the Verification 13

Reviewing the Verification Results 16

4 Conclusion 19

5 Appendix 20

Mapping between ISO/PAS 8800 Activities and CT Evidence 20

References 21

© 2026. Suresofttech Inc. All rights reserved.

1 Why Software Verification Matters for Automotive AI Safety

Software that uses AI can make wrong decisions. If a machine moves or a person takes an important action based on such a decision, the error is no longer just a wrong answer on a screen; it can lead to real harm. It is therefore necessary to check not only how accurate an AI decision is, but also what behavior it leads to.

On April 8, 2025, in Las Vegas, a Zoox robotaxi operating autonomously without passengers collided with a passenger car.[4] When the car quickly approached the robotaxi's lane from a commercial driveway, the robotaxi anticipated that the car would proceed forward, so it slowed down and steered to the right. Instead, the car came to a stop in the shoulder lane, and although the robotaxi braked hard, it could not avoid contact.[5, 6] In its recall report submitted to NHTSA, Zoox described the cause of the defect as follows: at speeds over 40 mph, the vehicle may make an inaccurately confident prediction of the other vehicle's behavior and be unable to avoid a collision.[4]

In this case, the wrong prediction led directly to driving decisions: deceleration and steering. This is because the model output becomes the input to subsequent motion planning and control. When verifying automotive AI software, it is therefore necessary to examine both the conditions under which the model outputs wrong results and how the software that receives those results behaves.

On the model side, the first step is to evaluate whether the data used for training and evaluation sufficiently covers the situations the vehicle will actually encounter, and whether the model still recognizes and predicts with the target performance when conditions change. This makes it possible to identify the conditions under which the model can be trusted and the situations that require additional measures.

© 2026. Suresofttech Inc. All rights reserved.

Knowing in which situations the model makes wrong predictions does not by itself reduce the risk of the system. The outcome of the same prediction depends on the behavior chosen by the software that receives it. Whether to use a prediction as it is, to restrict the behavior, or to switch to a fallback behavior is decided at the design stage, taking the known limitations of the model into account. In the Zoox case, the wrong prediction was output in the same way as an accurate one.[4] A wrong prediction that looks accurate is difficult to identify as wrong from the output alone. Therefore, a response that restricts or replaces the output, rather than using it as is, is needed. As an example of AI controllability, ISO/PAS 8800 describes a case in which, before the pulse-width modulation (PWM) signal of an actuator determined by an AI model is set, the PWM output is limited by a simple threshold or the consumer substitutes the PWM signal with an approximate physical model.[1, 3.1.2] Such responses are non-AI software logic. It must therefore be tested whether the actual code performs the response defined at the design stage when it receives normal and abnormal outputs.

Verification does not end with the behavior of individual code. It must also be confirmed that the intended response takes place while the model output is transferred, processed, and turned into system behavior, and testing must be repeated after the model or software is modified. Zoox also confirmed the root cause, implemented a software update, tested it in simulation and on test vehicles, verified it through its safety clearance process, and released it to all impacted vehicles operating on public roads.[4] The verification results produced at each stage of development and change become evidence that supports the safety requirements.[1]

This whitepaper describes how to verify automotive AI software and obtain safety evidence from the perspective of verifying a model together with the software that processes its outputs. Chapter 2 reviews the relationship between ISO/PAS 8800 and existing standards, and summarizes what must be checked and what evidence is needed for data, models, surrounding software, integrated systems, and operation and change. Chapter 3 describes how to turn the requirements of the surrounding C/C++ software into CT tests, and how to review execution results and coverage and link them to safety evidence. In this whitepaper, surrounding software refers to the C/C++ software that implements preprocessing, the inference interface, postprocessing, and safety mechanisms around an AI model. In ISO/PAS 8800 terms, it corresponds to AI components that are not AI models (preprocessing components, postprocessing components, and conventional software components) and to elements outside the AI system that limit the AI output.[1, 3.1.1, 3.1.2]

© 2026. Suresofttech Inc. All rights reserved.

2 ISO/PAS 8800: Explaining AI Safety with Evidence

Scope and Relationship to Existing Standards

ISO/PAS 8800:2024 applies to safety-related systems that include one or more electrical and/or electronic (E/E) systems that use AI technology and that are installed in series production road vehicles. The risks it addresses are those in which output insufficiencies, systematic errors, and random hardware errors of AI elements lead to undesired safety-related behavior at the vehicle level. [1]

In ISO/PAS 8800, functional safety risks caused by malfunctions of an AI system are addressed by tailoring or extending relevant clauses of the ISO 26262 series, and risks caused by functional insufficiencies are addressed by extending the concepts and guidance of ISO 21448 (SOTIF). However, because ISO 21448 applies to functions for which situational awareness obtained through sensors and processing algorithms is essential to safety, some systems that use AI may fall outside the scope of ISO 21448. For example, ISO 21448 does not apply to an engine controller that uses AI for performance optimization, but ISO/PAS 8800 does. It is therefore necessary to determine which standards apply to the target system and to link the activities of the applicable standards to one another.[1–3]

Required Verification and Safety Evidence

Explaining the safety of an AI system requires checking not only model accuracy but also the suitability of the data, the behavior of the surrounding software, system integration, and operating conditions. ISO/PAS 8800 addresses the dataset lifecycle in Clause 11, and verification against requirements together with AI component and integration testing in Clause 12. Clause 14.4.1 explains that operational risk can increase when operating conditions change or updates are applied. In addition, according to Clauses 8.3.1–8.3.2, lifecycle work products are used as the basis of the argument that the safety requirements are fulfilled.[1]

The following table summarizes what must be checked and what evidence must be obtained for each verification object.

© 2026. Suresofttech Inc. All rights reserved.

Table 1. What to check and what evidence to obtain for each verification object

Verification object	What to check against the requirements	Evidence to obtain
Data	Does the data reflect the defined operating conditions and hazardous scenarios? Are the sources, labels, quality, and selection criteria appropriate?	Data requirements, dataset identifiers and versions, quality and representativeness evaluation, exclusion conditions and limitations
Model	Does the model meet its performance and robustness targets under the defined conditions? What are its known limitations?	Model identifiers and versions, evaluation data and criteria, results and failure analysis per condition, scope of application
Surrounding software	Do preprocessing, the inference interface, postprocessing, and safety mechanisms behave as required?	Tests linked to requirements, inputs and expected values, execution and diagnostic results, structural coverage
Integration and system	Do the interactions between elements and the safety responses meet the system requirements?	Safety design and analysis, integration and system tests, simulation and HIL evaluation results where needed
Operation and change	Can the existing evidence still be used after new conditions or changes?	Field information, change impact analysis, re-verification scope and results, review and approval records

Data and model evaluations are also linked to the safety requirements. Data-related requirements serve as the criteria for collection, selection, verification, and maintenance, and model-related requirements are refined into evaluation conditions and pass/fail criteria. Dataset and model identifiers and versions, evaluation conditions, and limitations must be recorded to determine which configuration a result applies to.

© 2026. Suresofttech Inc. All rights reserved.

Model evaluation results must be distinguished from the safety validation results of the overall system. ISO/PAS 8800 calls the activity of evaluating the performance of an AI model candidate through testing AI model validation, and distinguishes it from validation in system development and from safety validation in ISO 26262.[1, 3.1.8] Each piece of evidence supports a different scope. Test sets and data suitability evaluations, model evaluation results and pass/fail criteria, and review records support whether the model achieves its target performance under the defined operating conditions and hazardous scenarios. Test results of the surrounding software linked to requirements show whether the designed responses are performed in the code, and structural coverage shows the range of code executed by those tests. Validation in system development and safety validation are performed separately, based on these results.

The assurance argument explains which safety claims and requirements these results support. According to Clause 8 of ISO/PAS 8800, it addresses not only the evidence but also the assumptions, limitations, and rationale for decisions.[1] In addition to whether individual tests passed, the argument examines whether the results come from the same item configuration and whether any parts still need additional verification. Only then can it be determined whether the results actually support the safety claims.

Among the verification objects above, CT provides test results for the surrounding software and the integration boundary. ISO/PAS 8800 states that AI components that are not AI models or do not contain AI models are not developed according to the standard, but their integration with AI components that are or contain AI models is.[1, 3.1.1] The unit testing and software integration testing of the surrounding software can therefore use the requirements-based testing and structural coverage that have been used in functional safety projects. In addition to these tests, CT provides results of testing the integration boundary with specified inference results in place of the actual AI model. The integration verification combined with the actual AI model (Clause 12.5.4), data and model suitability evaluation, safety design and analysis, and system-level verification are performed by the development organization, which links their results with the CT results. The next chapter describes this software verification process in detail.

© 2026. Suresofttech Inc. All rights reserved.

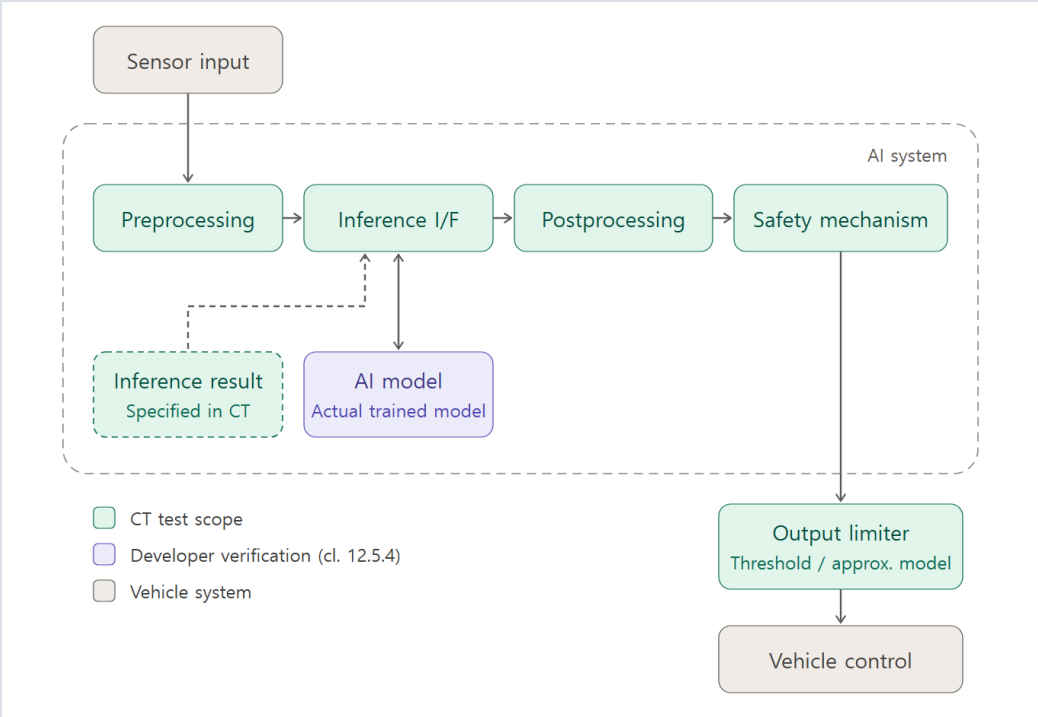


Figure 1 AI system structure and CT test scope

© 2026. Suresofttech Inc. All rights reserved.

3 Software Verification Evidence Obtained with CT

CT tests the requirements allocated to the software surrounding an AI model and provides a verdict per requirement together with structural coverage. In the preparation phase, the requirements are turned into CT tests; in the execution phase, the actual behavior of the code and diagnostic information are collected; and in the result review phase, the verdicts and coverage are used to find areas that need additional testing.

Table 2. CT features and work products by verification phase

Phase	CT features used	Work products
Preparation	Importing requirements from CSV files or ALM (application lifecycle management) tools, defining test cases, inputs, and expected values, configuring stubs	Tests linked to requirements
Execution	Running tests on host and target, reproducing normal and abnormal conditions with input settings, stubs, or fault injection, regression testing integrated with CI/CD environments	Actual values, PASS/FAIL, diagnostic results, execution logs
Result review	Verdicts per requirement, measuring structural coverage (statement, branch, MC/DC, function, call), exporting requirement-linked results to ALM tools	Verdicts per requirement, coverage and uncovered items

Preparing the Verification Environment

First, define the scope of the C/C++ code to be tested and its boundary with the AI model, and identify the CT version and features, the host and target integration configuration, source and build identifiers, and compiler options. This information serves as the basis for reproducing results and determining their scope of application.

© 2026. Suresofttech Inc. All rights reserved.

Before using CT for verification, determine the tool confidence level (TCL) in accordance with Clause 11 of ISO 26262-8:2018, and qualify the tool using methods appropriate to that level and to the target ASIL. A NOTE to Clause 15.3.2 of ISO/PAS 8800 also states that ISO 26262-8:2018, Clause 11 can be used to demonstrate confidence in the use of software tools.[1, 2] CT is a testing tool used in the development of safety-related software and is certified by TÜV SÜD in accordance with ISO 26262 and IEC 61508. The tool classification and usage restrictions according to ISO 26262-8 are described in the CT Safety Manual (see the "CT certification information" table below).[7] Review whether the scope of the CT Safety Manual and the certificate matches the version, features, and conditions of use actually applied, and record the judgment and its rationale.

Table 3. CT certification information

Item	Certification information	Source
Certification body	TÜV SÜD Product Service	Certificate[9]
Scope of certification	Qualified as a tool for use in the development of safety-related software according to ISO 26262, IEC 61508, EN 50716, IEC 62279, and IEC 60880; suitably validated for use in safety-related development according to IEC 62304	Certificate[9]
Test standards	ISO 26262-8:2018, IEC 61508-3:2010, EN 50716:2023, IEC 62279:2015, IEC 60880:2006, IEC 62304:2006+A1:2015	Certificate[9]
Tool classification (ISO 26262-8)	Tool impact TI2, tool error detection TD3	CT Safety Manual[7]
Applicable integrity level (ASIL)	All ASILs of ISO 26262:2018	Test report of the certificate[9]

© 2026. Suresofttech Inc. All rights reserved.

The verification engineer imports the software safety requirements into CT through CSV file import or integration with ALM tools (Codebeamer, Polarion, V-SPICE) supported by the applied version, and defines test cases, inputs, expected values, and pass/fail criteria for each requirement. The CT requirement view shows requirements that have no linked tests and requirements that need review because their content changed after import. The link status between requirements and tests can be exported as a requirement test coverage report.[8] If the model output characteristics assumed by a requirement or the test inputs were taken from a dataset or a model evaluation result, that source is also linked to the requirement and the test. The development organization reviews the completeness of the requirements themselves and the appropriateness of the pass/fail criteria.

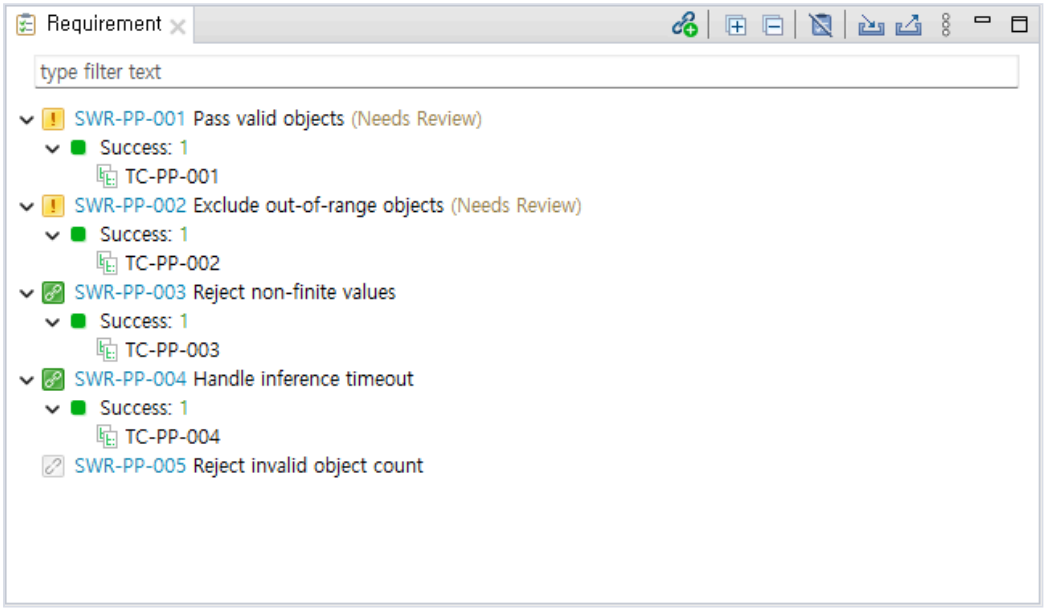


Figure 2 The requirement view shows requirements without linked tests and requirements that need review

Running the Verification

The verification engineer runs the prepared tests in CT. CT executes the input checks, data conversion, output validation, and error handling code of preprocessing, postprocessing, and the inference interface, and compares expected values with actual values to produce a verdict.

© 2026. Suresofttech Inc. All rights reserved.

The values returned by inference functions or external dependencies can be set with CT's input value settings, stubs, and fault injection, so normal and abnormal model outputs can be reproduced repeatedly. Diagnostic information and the safety state are specified as observation items to confirm that the safety mechanisms respond as designed. The conditions to reproduce and the expected responses are derived from the requirements and the safety analysis. What this testing confirms is the behavior of the surrounding software that receives the specified outputs; the performance and robustness of the model itself, and its handling of out-of-distribution (OOD) inputs, are confirmed in model evaluation.

The following are examples of tests that can be built by specifying the inference results of a perception model. The actual conditions and expected responses are defined by the project's requirements and safety analysis.

Table 4. Examples of tests with specified inference results

Output specified as the inference result	Observation items	Example of expected response
Object list within the defined range	Postprocessing result	Pass the postprocessing result to motion planning as input
Coordinates or confidence outside the defined range	Diagnostic information, postprocessing result	Exclude the object and record diagnostic information
Abnormal numeric values such as NaN	Diagnostic information, safety state	Reject the input and switch to a fallback behavior
Inference timeout	Safety state	Switch to a degraded state

© 2026. Suresofttech Inc. All rights reserved.

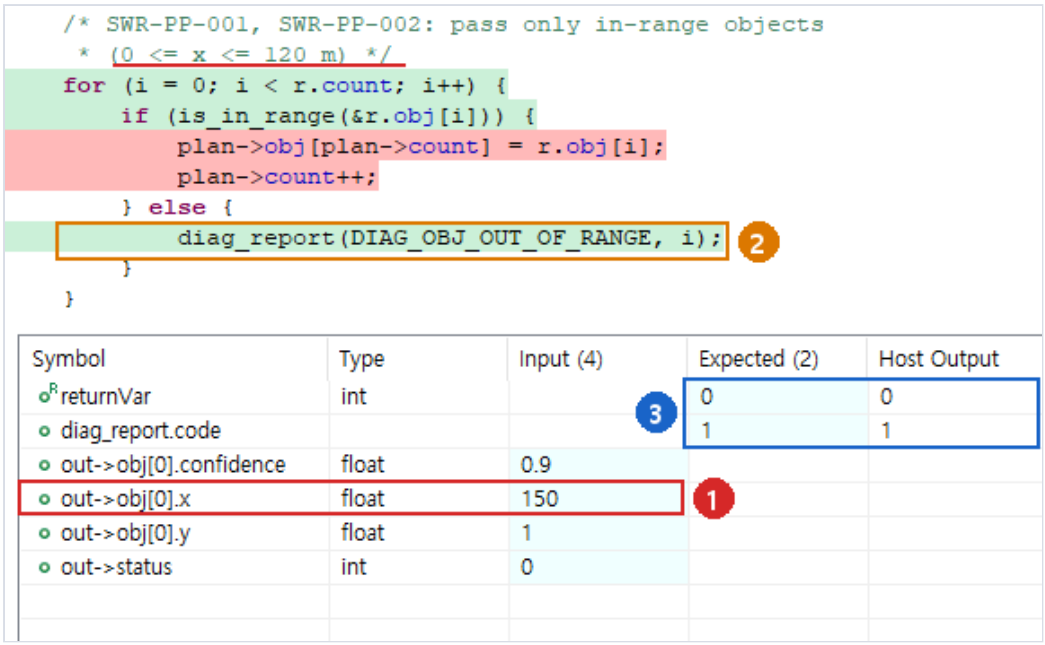


Figure 3 When the infer_get_result() stub returns a coordinate outside the defined range (①), pp_process_frame() excludes the object and records diagnostic information (②, ③)

Coverage is measured by inserting instrumentation code (probe code) into the source code under test. Because instrumentation is inserted only into the code under test, the execution time of uninstrumented libraries such as an inference engine does not change even when they run together. How much the size and execution time of the code under test increase depends on the coverage metric and the code structure. According to NOTE 4 of Clause 9.4.4 of ISO 26262-6, when coverage is measured with instrumented code, evidence that the instrumentation did not affect the test results may be needed; this evidence can be shown by re-running representative test cases on uninstrumented code and comparing the verdicts.[2] Timing characteristics such as the execution time from preprocessing to postprocessing and deadlines are checked on the target environment with an uninstrumented build.

The development and deployment environments can differ in compiler, optimization options, numeric representation, and hardware. According to Clause 9.4.5 of ISO 26262-6, when software unit testing is not performed in the target environment, the differences between source and object code and between the test and target environments are analyzed, and additional tests to be applied in the target environment in later test phases are determined from that analysis.[2] Annex C of ISO/PAS 8800 (ISO 26262 gap analysis for ML) also addresses this clause. CT runs tests repeatedly on the host and, in target configurations integrated with supported major debuggers, runs the same test assets on the target where possible, providing actual values, verdicts, and coverage for each environment. With these results, the verification engineer checks whether the state transitions and diagnostic information in both environments match the expected behavior, and compares numeric results against predefined tolerances. The results of the difference analysis are recorded together with the configuration information.

© 2026. Suresofttech Inc. All rights reserved.

When a test fails, CT provides the requirement of the failed test, the inputs, expected and actual values, the execution log, and the range of executed code. Based on this information, the verification engineer classifies the cause as code, requirement, or environment. Code issues are re-verified by CT regression testing with the same tests after the fix; for requirement issues, the requirements or pass/fail criteria are revised; and for environment issues, the test configuration is improved before testing again. If it turns out that the model output characteristics assumed by a requirement may differ from the actual model, this is handed over to the model and data evaluation and safety analysis activities.

When new conditions are found during operation, or when the model, software, or deployment configuration changes, the development organization evaluates the safety impact and defines the re-verification scope. Field monitoring and change impact analysis are activities of the development organization, and CT is used for the re-verification defined by the impact assessment. The same applies when the model is retrained and redeployed. Clause 14.8.3 of ISO/PAS 8800 addresses AI re-training, re-validation, re-approval, and re-deployment. The verification engineer adds the defined conditions to the existing test assets and performs regression testing, and CT provides evidence for comparing execution and coverage results before and after the change. Because CT can be integrated with CI/CD environments through interfaces such as its command line interface and Jenkins plugin to automate test execution and report generation, the same regression tests can be repeated for every change. In addition, by importing into CT the coverage obtained in integration and system testing with COVER, Suresofttech's system coverage measurement tool, only the code not executed in that testing can be selected and tested again with CT.[8]

Reviewing the Verification Results

Result review starts by checking the verdicts per requirement and the structural coverage in CT together. Even if all tests pass, some response paths may remain unexecuted. After defining the coverage targets and goals, the execution range is measured with the metrics supported by the applied CT version, such as statement, branch, MC/DC, function, and call coverage.

© 2026. Suresofttech Inc. All rights reserved.

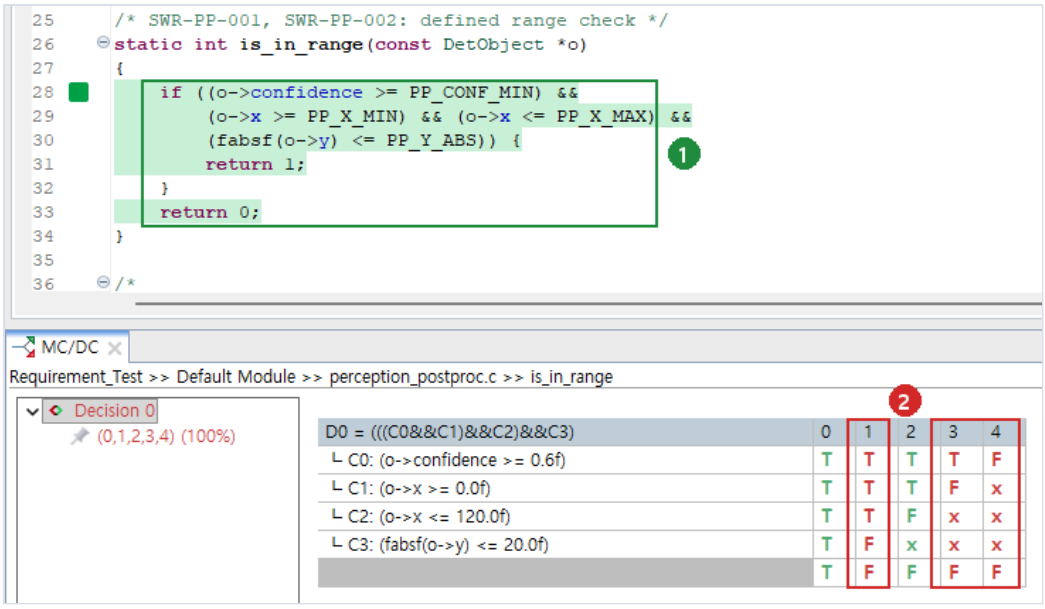


Figure 4 All code of is_in_range() was executed (①), but MC/DC confirmed the independent effect of only one of the four conditions (②)

The verification engineer reviews the uncovered items to determine whether additional inputs or abnormal-condition tests are needed. Uncovered items include not only code that was not executed, but also conditions that were executed but whose independent effect on the decision outcome was not demonstrated in MC/DC. Code that cannot be executed or that was excluded from the scope is linked to the project's justification records.

For the CT work products reviewed in this way to serve as evidence in the assurance argument, they must be linked to the requirements they support, the configuration in which they were produced, and the person who reviewed them.

© 2026. Suresofttech Inc. All rights reserved.

Table 5. Information to link to the assurance argument

Category	Key information	Role in the assurance argument
CT test assets	Software requirements, test cases, inputs and expected values, error conditions	Reproduce the software test conditions and pass/fail criteria
CT execution and coverage work products	Actual values, PASS/FAIL, diagnostic results, execution logs, structural coverage and uncovered items	Present the behavior and test range per requirement
Safety design and system verification	Hazard and requirement allocation, safety mechanism design, integration and system test results	Link the rationale between individual code tests and system safety responses
Configuration information	Configuration identifiers identified when preparing the verification environment, change history	Identify the object and point in time to which the evidence applies
Project review information	Justification of exclusions and uncovered items, difference analysis, reviewers, approval status, known limitations	Adopt the work products as evidence in the assurance argument

The development organization consolidates this information per requirement, reviews uncovered and excluded items and known limitations, and records why the test results support the requirements. Conditions that cannot be detected by CT testing and system behavior that has not been confirmed remain subject to further evaluation. The CT work products prepared in this way support the safety claims together with the data and model evaluations and the integration and system verification described in Chapter 2.

CT test work products can be used as software verification input to the AI system verification report of Clause 12.6.1 of ISO/PAS 8800. The development organization is responsible for structuring and approving the verification report and the final assurance argument.

© 2026. Suresofttech Inc. All rights reserved.

4 Conclusion

The key to addressing ISO/PAS 8800 is to link the verification results of data, models, and software to the safety requirements, and to support the safety claims with reviewable evidence based on that link.

Among these, CT produces verification work products for the software and the integration boundary that connect the AI model to vehicle functions. It links requirements to test cases, specifies and repeatedly reproduces normal and abnormal outputs of the AI model, and presents the execution results and structural coverage of preprocessing, postprocessing, the inference interface, and safety mechanisms. Because CT is a testing tool certified in accordance with ISO 26262, the CT certificate can be used as the basis for tool confidence once it is confirmed that the certification scope matches the actual conditions of use.

When CT work products are linked to configuration identification and review and approval records, they become traceable and reviewable software evidence, and can be used together with data and model evaluations and safety design and system verification as the basis of ISO/PAS 8800 safety claims.

© 2026. Suresofttech Inc. All rights reserved.

5 Appendix

Mapping between ISO/PAS 8800 Activities and CT Evidence

Table 6. Mapping between ISO/PAS 8800 activities and CT evidence

ISO/PAS 8800 area	Activity in the standard	CT contribution	Key evidence	Scope of application
Clause 8	Structuring the relationship between safety claims and evidence	Provides links between requirements, tests, and results	Traceability information, test results, coverage	Software verification claims
Clause 9	Derivation and refinement of AI safety requirements	Links requirements allocated to software to tests	Tests and verdicts per requirement	The development organization reviews the completeness of the requirements
Clause 12	Verification and validation of the AI system, work products	Tests the surrounding C/C++ software and the integration boundary	Test specifications, execution results, coverage	Model performance validation and system validation are performed as separate activities and presented together with CT results
Clause 13	Analysis of AI errors, their causes, and safety impact	Reproduces observable conditions identified in the analysis	Error condition setup and safety response results	Completeness of the error list is addressed by the safety analysis
Clause 14	Re-evaluation during operation and update of evidence	Regression testing after changes	Execution and coverage results per version	Re-verification scope is determined by impact analysis
Clause 15	Confidence in use of AI development frameworks and software tools used for AI model development	Presents the certificate and safety manual according to ISO 26262-8 as a software verification tool	Certificate, safety manual, tool confidence analysis	The CT version and conditions of use must match the certification scope and the conditions of use in the safety manual
Annex C	ISO 26262 gap analysis for ML	Measures coverage of the	Structural coverage and	An informative annex; not a basis for applying

© 2026. Suresofttech Inc. All rights reserved.

References

- [1] ISO/PAS 8800:2024, *Road vehicles — Safety and artificial intelligence*. Official information.
- [2] ISO 26262:2018, *Road vehicles — Functional safety*. Software verification: Part 6; tool confidence, configuration and change management: Part 8.
- [3] ISO 21448:2022, Road vehicles — Safety of the intended functionality.
- [4] Zoox, Inc., Part 573 Safety Recall Report 25E-029, submitted 2025-05-01, pp. 1–2. Manufacturer's report on the 2025-04-08 collision and the prediction error; it is not a final accident investigation report independently produced by NHTSA.
- [5] Zoox, Inc., Zoox issuing voluntary software recall, 2025-05-06. Manufacturer's announcement on the circumstances of the 2025-04-08 collision.
- [6] TechCrunch, Amazon-owned Zoox issues recall following robotaxi crash, 2025-05-06.
- [7] Suresofttech Inc., *CT Safety Manual*. Online manual.
- [8] Suresofttech Inc., *CT User Manual*. Online manual.
- [9] TÜV SÜD Product Service GmbH, *Certificate No. Z10 126942 0001* (Software Tool for Safety Related Development, CT).

The descriptions of the standards summarize the original texts and are neither official translations nor direct quotations.

© 2026. Suresofttech Inc. All rights reserved.