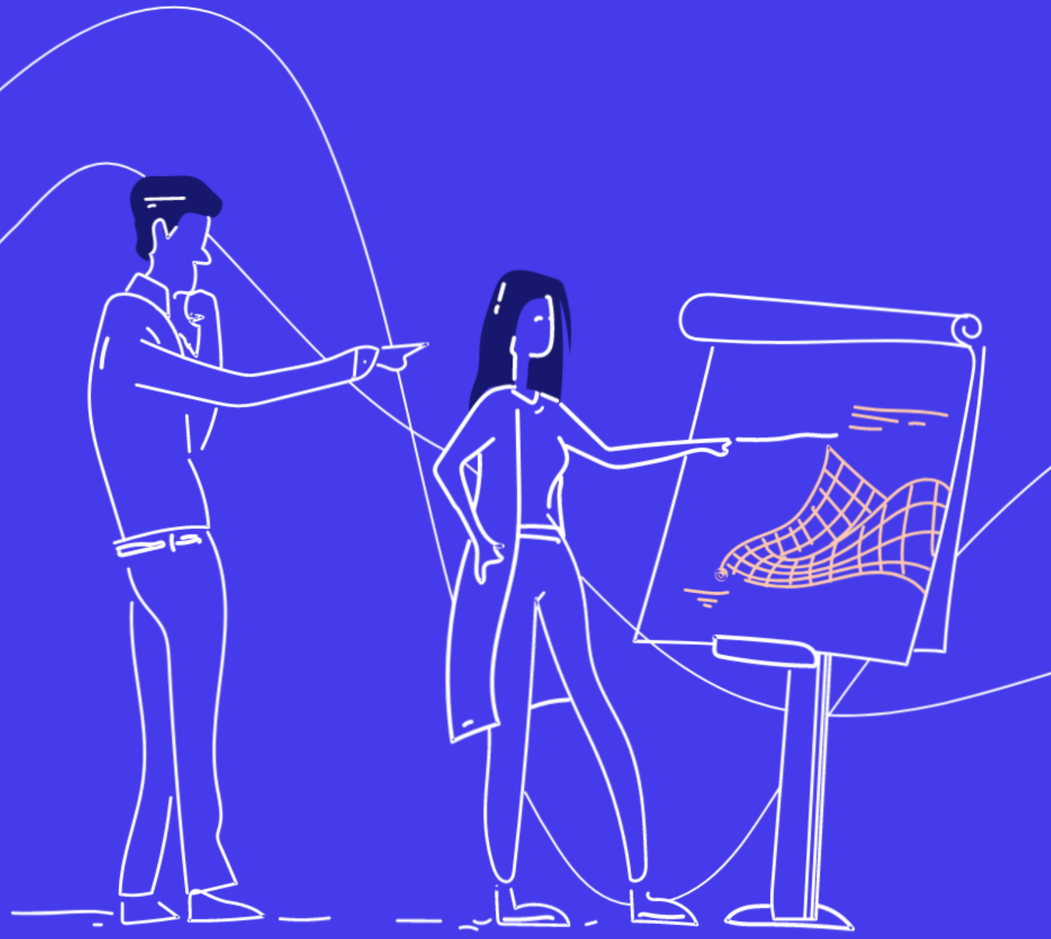
sciengineer

Polyspace for C/C++ Code Verification



SciEngineer's training courses are designed to help organizations and individuals close skills gaps, keep up-to-date with the industry-accepted best practices and achieve the greatest value from MathWorks® and COMSOL® Products.

Polyspace for C/C++ Code Verification

This two-day course discusses the use of Polyspace Code Prover to prove code correctness, improve software quality metrics, and ensure product integrity. This course describes techniques for creating a verification project, reviewing and understanding verification results, emulating target execution environments, handling missing functions and data, managing unproven code, applying MISRA-C rules and reporting analysis results.

Prerequisites

Strong knowledge of C or C++

TOPICS

Day 1

- Polyspace Workflow Overview
- Polyspace Bug Finder Analysis
- Analyzing Polyspace Code Prover Results
- Code Verification Checks

Day 2

- Managing Polyspace Code Prover Verifications and Results
- Adding Precision to Polyspace Code Prover Verifications
- Integration Analysis
- Application Analysis

Polyspace Workflow Overview

OBJECTIVE: Become familiar with Polyspace Bug Finder and Polyspace Code Prover and work through an introductory example.

- Software development workflows with Polyspace
- Simple verification example
- Analyzing defects and run-time errors

Polyspace Bug Finder Analysis

OBJECTIVE: Analyze code that may not be ANSI C compliant and account for the runtime environment, and correct defects and coding rule violations using Polyspace Bug Finder.

- Common run-time environment artifacts
- Handling processor-specific code
- Defining the execution context
- Setting target hardware information
- Analyzing and managing Polyspace Bug Finder defects
- Detecting coding rule violations
- Measuring code metrics

Analyzing Polyspace Code Prover Results

OBJECTIVE: Become proficient at interpreting Polyspace Code Prover results.

- Overview of abstract interpretation
- Call tree analysis
- Source code navigation
- Execution paths
- Variable ranges
- Global variables

Code Verification Checks

OBJECTIVE: Find run-time errors using diagnostics available in Polyspace Code Prover.

- Overview of C source code checks
- Location of checks in source code
- Description of checks
- Relevant verification options

Adding Precision to Polyspace Code Prover Verifications

OBJECTIVE: Learn how Polyspace Code Prover treats missing code during verification, and how to affect this behavior to produce more meaningful verifications.

- Robustness verification and contextual verification
- Function stubbing
- Data range specification
- Manual stubbing

Integration Analysis

OBJECTIVE: Learn how to manage verifications with increasing code complexity, and how to interpret and compare integrated analysis with robust analysis.

- Managing code modules
- Analyzing integration defects and rule violations with Polyspace Bug Finder and Polyspace Code Prover
- Importing comments

Application Analysis

OBJECTIVE: Review procedures and options that are useful when verifying complete applications.

- Setting up an application verification
- Improving the results of an application verification
- Detecting concurrency issues
- Comparing robustness and contextual verification
- Creating documentation

Managing Polyspace Code Prover Verifications and Results

OBJECTIVE: Handle verification results that contain large amounts of unproven checks.

- Determining verification effort
- Performing a quick review
- Performing a selective orange review
- Setting verification precision
- Prioritizing orange checks
- Reviewing orange checks

sciengineer

Expand your knowledge