Polyspace for C/C++ Code Verification

Prerequisites

Strong knowledge of C or C++

Day 1 of 3	
Polyspace Workflow Overview	**Objective:** Become familiar with Polyspace Bug Finder and 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 run-time environment, and correct defects and coding rule violations using Bug Finder. Common run-time environment artifacts Handling processor-specific code Defining the execution context Setting target hardware information Analyzing and managing Bug Finder defects Detecting coding rule violations
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

Day 2 of 3

Day 2 of 3	
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
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 Bug Finder and Code Prover Importing comments
Measuring and Reporting Software Quality	**Objective:** Use Polyspace metrics to share and catalog verification results, and generate standard reports from verification results. Sharing results Using Polyspace metrics Testing software quality objectives Creating documentation
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

Day 3 of 3

Day 3 of 3	
Hands-on Instruction (Optional)	**Objective:** Spend time reviewing what you have learned and applying Polyspace Code Prover directly to your own project. Potential topics include: Bug Finder checks Verifying C++ code Tasking and shared data analysis Verifying generated code Development process review Workflow integration Client/server software installation Polyspace configuration for project code Results interpretation