# Testing and Simulation with PTC Perc® Real-Time Java®


## Introduction

This White Paper describes the benefits of using PTC Perc real-time Java for Testing and Simulation of complex software-defined products.

## Testing Complex Software-Defined Products

Today's high-technology products, from smartphones to automobiles to unmanned aircraft, are increasingly "software defined." Their capabilities and features are determined largely by software rather than physical hardware. Even when additions and improvements are made to hardware, the end-user experience of those changes is often shaped by software. Higher resolution cameras are added to smartphones with each new model, but the "wow factor" comes from improved sensitivity, noise reduction, and other enhancements performed by software during image post-processing. Advanced Driver Assistance Systems (ADAS) use sensors to detect obstacles and identify driver errors but the AI-based decision making is done in software.

More software means more lines of code, which means more chances for programming errors to sneak past quality assurance processes and into the hands of a potentially unhappy customer. And in the case of safety critical systems, the customer's happiness is not the only concern.

When tens of thousands or millions of lines of code are involved, traditional quality assurance methodologies fall short. It takes too long to test all possible operational conditions, failure modes, edge cases, and rare events with

human-driven testing. Some conditions can be difficult to reproduce in a lab (anyone have an altitude chamber?). The sheer volume of test cases requires an automated process. In other words, we must use software to test software.

# Automated Software Testing Methodologies

Automated testing is most effective when it begins with Model-Based Design (MBD). Before mechanical, electronic, or software designs begin, MBD helps developers conceive and visualize the product, codify system requirements, and analyze implementation trade-offs. The models created become the foundational "source of truth" for subsequent planning, architecture, design, implementation, testing, and production phases of a product.

## Testing via Model Simulation

An important advantage of Model-Based Design is the "divide and conquer" approach that arises from it. Modeling the system allows developers to think about how to partition it into components and define internal states and data within each component as well as interactions among them. From this approach, hardware and software architecture can begin to emerge.

Many MBD tools provide simulated execution of models while logging results for analysis. "Model-in-the-Loop" (MiL) testing is a major feature of these tools. Developers can validate and refine their models long before hardware or software prototypes are designed and built.

## Code Generation and Software Simulation

After a period of MiL development to refine models and create test cases, MBD tools such as PTC's Modeler and Codebeamer products can generate code from models. This allows developers to move from pure model-based simulation to running software that implements those models in a programming language like Java. The benefits of using high-level Java rather than C or C++ are productivity and portability across platforms. Test Engineers can quickly build and run Java-based "Software-in-the-Loop" (SiL) simulations on desktops, servers, or embedded computers before target hardware becomes available. Communications with simulated or physical sensors, actuators, and other model components are exchanged over a network using low-level protocols or a publish/subscribe messaging system such as Data Distribution Service (DDS).

When a physical component of the system becomes available, developers can integrate with SiL components for "Hardware-in-the-Loop" (HiL) testing. This facilitates realistic stress testing of mechanical and electrical components before final down-selections.

The evolution of an automated simulation and testing environment from MiL to SiL to HiL and combinations thereof provide developers with opportunities to identify and correct errors and deficiencies in

requirements, models, architecture, hardware, and software early in the life cycle rather than postponing that effort until prototypes are available. It also provides time to create a comprehensive library of test cases and verification data to apply to the final product.

## Where Traditional Java Runs Out of Steam

While the Java language and ecosystem make for a highly productive and portable environment, traditional Java Virtual Machines have a weakness when it comes to simulated testing: they can't do real time. From unpredictable pauses for garbage collection and Just-in-Time (JIT) compilation to uncontrolled thread scheduling and page faults, there is no way a traditional JVM can guarantee it will meet timing deadlines. While running test cases that would normally fail if a software-simulated component doesn't respond within a deadline, traditional Java misses the mark. Developers are forced to settle for low-level C/C++ with its incumbent longer coding and debugging cycles, memory safety problems, and portability issues. This slows the process of building and maintaining a robust simulation and testing environment. But it doesn't have to be that way.

## Where PTC Perc Shines

PTC Perc is a Java Virtual Machine and developer tool chain purpose-built to bring all the benefits of the Java language to systems with sub-millisecond real-time latency requirements. It runs power plants, robotic oil rigs, and weapons control systems. And it can run Java-based Software-in-the-Loop simulations with deterministic timing.

How is that possible? A common belief among software developers is that Java cannot do real time. Just because Oracle Java and OpenJDK suffer from unpredictable pauses caused by garbage collection, JIT compilation, and non-deterministic thread scheduling doesn't mean every Java Virtual Machine must behave that way.

PTC Perc is designed for applications that require deterministic behavior. Always. Automatic garbage collection (GC) is key to Java's memory safety and Perc's GC can reclaim memory held by dead objects and compact live objects to make room for new allocations even while Java threads continue running. Perc's GC parallel worker threads run concurrently with Java threads and high-priority Java threads can preempt a GC worker at any time.

But there's more. Traditional JVMs interrupt Java programs to JIT-compile or optimize code at unpredictable times, delaying execution of that code. PTC Perc allows developers to compile Java code to native instructions Ahead-of-Time (AOT), bypassing the JIT compiler entirely and improving application startup time.

Traditional JVMs let the Operating System decide when to run Java threads. Yes, the Java libraries allow setting thread priority, but this is little more than a hint for scheduling. Operating System schedulers differ greatly and often attempt to "fairly" allocate CPU time based on a thread's usage history. This violates

fundamental real-time design principles where thread priority is sacrosanct. PTC Perc, on the other hand, has a built-in dispatcher that assigns Linux real-time scheduling policies, priorities, and CPU core affinities to running threads. It allocates available cores to threads on a strict priority basis. It also implements priority inheritance protocol to eliminate priority inversions among Java threads.

## It's Time to Consider a Better Way

Are you tired of chasing bugs and crashes in your C/C++ simulations when you could have spent that time running them? Isn't it time to build a modern testing and simulation environment using a memory safe, modular, scalable, and deterministic Java Virtual Machine? If you would like to learn more about what PTC Perc can offer, check us out at **ptc.com/perc**



695902_Testing_and_Simulation_with_PTC_Perc_Real-Time_Java_Whitepaper_05_25