PolySpace for C++

Do you want to reduce your coding and testing effort?

The industrial challenge

Embedded systems are at the heart of the world's most critical applications and assuring their reliability is of paramount concern. The increase in size and complexity of current applications results in rising testing costs. It also highlights the limitations of available tools and methods.

Indeed, when it comes to finding runtime errors, classical methods are inadequate. For example, tools that compute metrics or enforce the use of coding rules may help in introducing fewer bugs during the coding but will not actually find bugs. Dynamic testing, which is of undisputable value for functional validation, only catches anomalies when they surface during test case execution. Furthermore, with traditional approaches, every error detected during program execution requires hours, days, even weeks of debugging.

Today's applications require a new generation testing solution that is capable of improving software quality while reducing the usual costs associated with such efforts.

A new approach

PolySpace allows you to detect run-time errors and concurrent accesses on shared data automatically very early in the development process without testing.

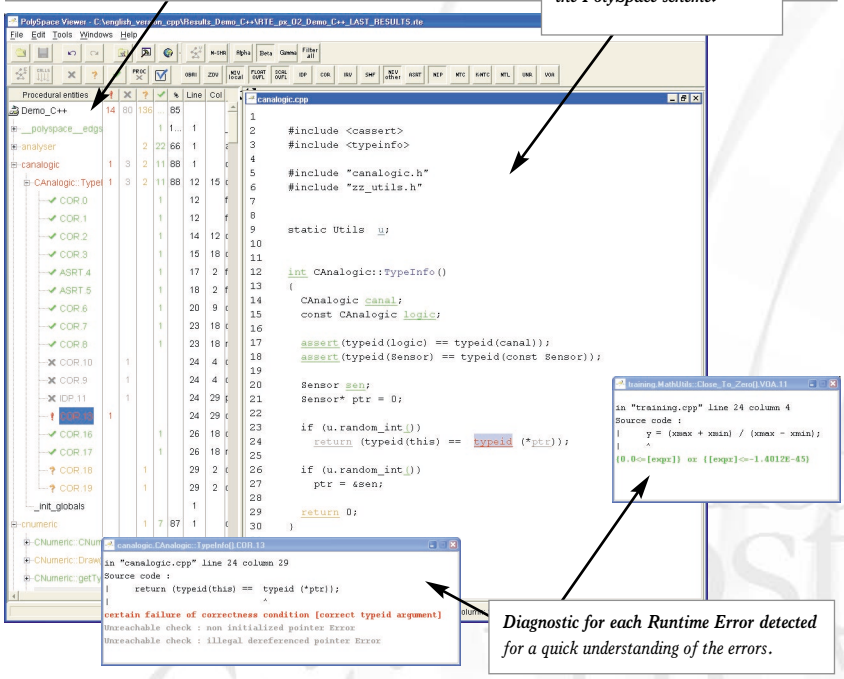
It statically analyses the dynamics of software applications by relying solely on the source code.

- **No test cases to write;**
- **No modification of the code;**
- **No application execution necessary.**

In contrast to dynamic testing, PolySpace automatically and directly highlights causes of run-time errors in the source code (therefore, no debugging).

The net result is a drastic reduction in software development costs by dramatically reducing the need for testing and debugging.

List of operations causing ***Runtime Errors*** *with macro expansion and filters for quickly focusing on the most critical errors. Each operation checked is displayed using a meaningful colour scheme and related diagnostic:*
Red Errors which occur at ***every*** *execution,*
Green Error conditions that will ***never*** *occur,*
Orange Warning – an error may occur **sometimes,**
Gray Shows unreachable ***(dead)*** *code.*



Runtime Errors detected

Run-time errors are faults whose consequences include processor halt, data corruption or security breaches. They are latent faults that generally surface during execution. They have a major business impact: testing goes over budget, delayed releases, service disruptions during operation (e.g. sending uncontrolled commands to external devices).

Errors detected by PolySpace include:

- **Read access to non-initialized data** (variables and function return values);
- **De-referencing through null** and **out-of-bounds pointers;**
- **Out-of-bounds array access;**
- **Invalid arithmetic operations** such as division by zero, sqrt(negative number);
- **Overflow / underflow** on arithmetic operations for integers and floating point numbers;
- **Dangerous type conversions** e.g. long → short, float → int;
- **Access conflicts for data shared between threads;**
- **Invalid dynamic_cast calls;**
- **Throws of unauthorized exceptions;**
- **Calls to virtual pure methods;**
- **Negative size arrays;**
- **Null receivers;**
- **Null pointers to members;**
- **Wrong type for receivers** such as polymorphic type in non-virtual method;
- **Throws during catch parameter construction.**

PolySpace also detects:

- **Non-terminating function calls and loops;**
- **Unreachable code** (dead code).

Static Verification

Static verification of dynamic properties is a well-established method for representing all possible executions of a given software application, based solely on the source code.
The basic idea is to calculate the domain of values each variable can take, for each point in the application by applying multi-thread, interprocedural and control structures analysis. This technique is used by PolySpace to identify run-time errors. This automatic approach makes PolySpace the unique testing tool that can statically predict which run-time errors will affect your application prior to tests and deployment. e.g.,

```
int  tab [100] ;
int  *p = tab ;
int  i ;
for  (i = 0 ;  i < 100 ;  i++, p++)  *p = 0 ;
*p = 5 ;
```

The expression `*p = 5 ;` causes a memory corruption that is automatically detected by PolySpace. All other testing methods would request human efforts to catch it (code review, test execution or code instrumentation).

Supported C++ Language Features & Software Engineering Standards

PolySpace supports C++ as defined in the ISO/IEC 14882:1998 standard. Nested classes, polymorphism, templates, multiple inheritance and other C++ specific features are supported by PolySpace. It allows developers to benefit from both object-oriented programming facilities (reusability and easier maintenance of code) and best-of-class analysis tools.
Furthermore PolySpace tools naturally integrate with software engineering standards such as **MISRA, DO-178B, IEC 61508, CENELEC, FDA, TickIt, CMM, or IEEE 1012.** They are a means to automate or improve several activities prescribed by these standards such as code evaluations, static analysis, control/data flow analysis, providing definite transition criteria and avoidance of runtime errors.

Benefits

PolySpace brings both quality and productivity improvements for developers of embedded applications as well as for quality assurance engineers.

PolySpace Developer Edition

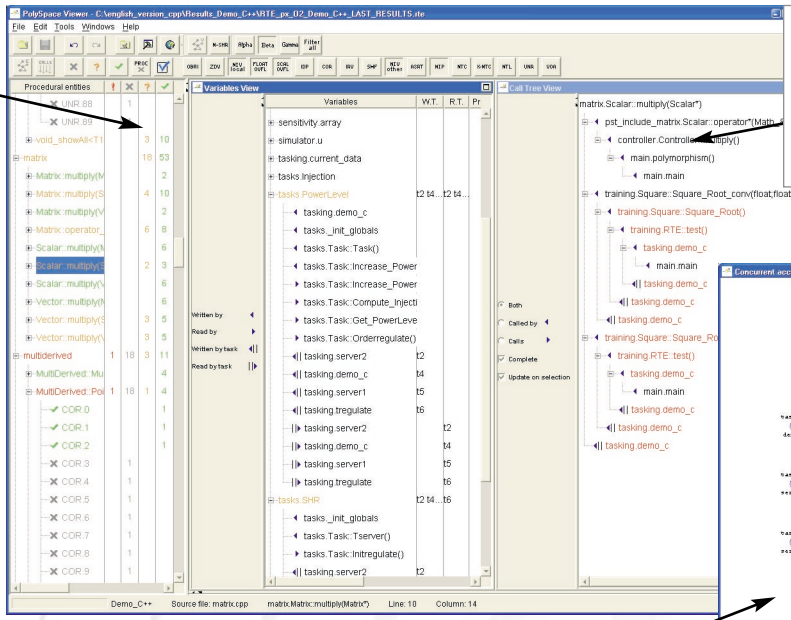
Developers benefit from an early detection of errors – before any testing – which is the key and crucial element to reducing testing and debugging times. PolySpace offers the earliest and first automatic detection of run-time errors at compilation time. With no test cases to write, no instrumentation or execution of the application, PolySpace produces immediate savings. The errors that PolySpace finds early in development would involve appreciable debugging time if detected later during the test phase. Furthermore, performing functional tests on a module analysed by PolySpace becomes much easier thanks to more robust source code. So, why spend time debugging runtime errors when PolySpace would have detected them earlier and automatically?

PolySpace Auditor Edition

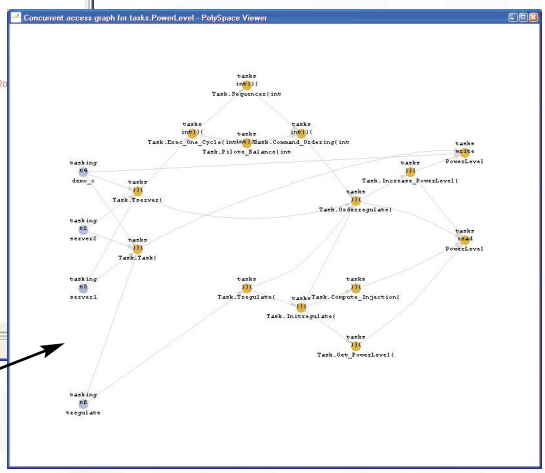
Quality Assurance Engineers get a reliable assessment of the quality of software applications, which is a key issue in today's supplier/end user relationship. PolySpace offers a way to improve the quality of software applications automatically, by drastically reducing the number of bugs that can surface during late system test after code freeze as well as at the end-users site. PolySpace provides fast, repeatable and low cost acceptance criteria for software applications. It is suitable for both internal quality control performed by an independent validation and verification team as well as to assess outsourced development and test activities.

PolySpace finds more bugs than classical testing & much faster.

***A complete and interactive Call Tree** is provided for documentation and navigation purposes. Fully exportable to Excel © format.*



***Global Data Dictionary** including read and write access to all global data (pointer and alias included). It also identifies data shared between threads and related protection mechanisms (critical sections, temporal exclusions). Fully exportable to Excel © format.*



***Access Graph for each Shared Data item:** A meaningful and efficient feature showing how threads access shared data through function calls.*

PolySpace TECHNOLOGIES

Email: contact@polyspace.com
www.polyspace.com