DTP 5.3.0 and DTP Engines 10.3.0 (Jtest, C++Test, dotTest)

The latest release of Parasoft Development Testing Platform (DTP) and DTP Engines for C/C++, .NET, and Java build on Parasoft's innovative approach to continuously improving software quality processes. This release includes new features that provide deeper visibility into how changes in the code affect risk, a new unit testing interface for Java developers that will improve the way tests are created and maintained, and enhanced support for automotive software quality. We've also enhanced UIs, updated widgets, improved the award-winning Process Intelligence Engine (PIE), and more.

In this release:

- Understanding Change in Continuous Delivery Environments
- A Better Desktop Experience for Developers
- Extended Support for Automotive Software Quality
- Other Updates and Enhancements
- New and Updated Code Analysis Rules

Understanding Change in Continuous Delivery Environments

As organizations implement continuous software delivery platforms, conventional reporting mechanisms based on data aggregated over time are no longer viable. Understanding the risk that incremental changes introduce from build to build becomes critical as the release velocity accelerates. Organizations need immediate visibility into changes in coverage, changes in test regressions, and the ability to quickly identify which tests need to be rerun. The updates in this release help you understand these changes and identify any issues that need to be resolved in order to mitigate risk and accelerate delivery.

Viewing and Comparing Static Analysis Violations by Build

In this release, we've extended the concept of viewing and comparing data based on build so that you can immediately see deltas in new, fixed, and existing static analysis violations. This enables organizations to not only address potential defects more efficiently early in the SDLC, but also correlate violations with change to pinpoint specific areas of the code where defects may have been introduced.

Updates to Change-based Testing PIE Slices

Building on our application coverage technology, which enables you to combine test and coverage results across unit, functional and manual testing techniques, DTP has been updated to improve our change-based PIE Slices.

- **Change-based Testing**: This slice analyzes changes in the code base and correlates those changes with testing metrics to identify tests that require re-execution. The output of the analysis can be displayed in multiple forms (pie chart, table report, downloadable as CSV data and in the DTP Test Explorer) and reprioritizes retesting efforts based on the data and policy.
- **Risky Code Changes**: This slice calculates the risk associated with code changes based on Test Deficit, Maintenance Burden, and Quality Debt. The metrics are displayed in DTP in multiple forms (pie chart, bubble chart, heat chart, and table report).
- **Untested Changes**: This slice analyzes the coverage data for modified lines of code in a build verses a baseline and reports coverage gaps based on the change. This helps reduce the testing scope associated with code changes, such as implementing a new feature, so testers can focus on the uncovered code and increase their testing efficiency. It also helps teams maintain an audit trail of the risk associated with untested changes.

Visit the Parasoft Marketplace to download these PIE slices and other extensions for Parasoft solutions.

Coverage Agent Manager (CAM)

CAM provides a web interface for controlling the collection of coverage data and test results during test execution of manual tests of an application under test (AUT). In addition to the web interface, CAM provides a REST API that enables you to monitor code coverage and test data integration from third party test frameworks, such as Selenium.

Visit the Parasoft Marketplace to download the Coverage Agent Manager, as well as extensions and integrations for CAM that leverage the API.

Multisession Application Coverage for Java Web Applications

The Java application coverage engine (Jtest DTP Engine) has been extended to track the unique coverage for users that are simultaneously accessing the same application server. Multiple QA engineers, for example, can access the same web application server, perform parallel testing, and the coverage agent will be able to determine which coverage is associated with which server.

A Better Desktop Experience for Developers

Developers want to write code, not be burdened with more tools and processes that slow them down. This release introduces the following IDE-based features and enhancements enable developers to create defect-free software faster.

Unit Testing Assistant for Java

The shift to Agile has amplified the challenges associated with unit testing that threaten to erase the gains made by adopting iterative development approaches in the first place:

- Unit tests are expensive to create and maintain in terms of resources. Someone with the technical knowledge and understanding of the code is required to construct the tests and update them as the code evolves.
- Unit tests are often executed without the context of the user story. Without traceability from test to requirement, organizations may not have full confidence that the test verifies the intended functionality.
- Unmaintained or poorly constructed tests might behave inconsistently and become highly dependent on the environment. As a result, tests may introduce volatility into the regression suite as the code evolves, making it impossible to reliably execute the tests in a continuous environment.

The Jtest Unit Test Assistant (UTA)[a], which helps developers create meaningful, maintainable unit tests, solves these problems. UTA not only reduces the time to create unit tests, its interface removes some of the technical barriers that have historically required developer expertise.

In addition to easing the burdens of writing unit tests, UTA can actually analyze existing tests to root out potential issues, such as stability. The unit test monitoring functionality helps identify tests that are highly dependent on the environment, enabling teams to improve the test so that it can consistently run in isolation. Additionally, UTA enables a developer to track an object during test execution to generate assertions on the observed changes of the object.

Continuous Quality Assistant (CQA)

CQA enables developers to get immediate feedback on the code they're working with. CQA analyzes code as the developer performs actions, such as opening and saving files, prior to running a full analysis. CQA executes targeted, high-performance rules as controlled through the test configuration, keeping analysis overhead low.

Extended Support for Automotive Software Quality

As the focus on quality and safety-critical aspects of automotive software come to the forefront of the industry, Parasoft continues to expand its offering for C and C++ applications.

Complete MISRA C:2012 Coverage for Static Analysis

The static analysis rules shipped with the DTP Engine for C/C++ have been updated to include complete coverage for the MISRA C:2012 standard, including complete coverage for Amendment 1 published in May of 2016. See [link to coverage doc] for details.

The marketplace also includes a new MISRA qualification kit, which provides configurations for the DTP Engines and the DTP server for centralized reporting. The kit contains a set of regression tests and supporting documents to help you validate your use of MISRA C:2012, such as when applying static analysis rules for ISO 26262 compliance.

New Marketplace Extensions

In addition to the updates to the DTP Engine, we've introduced the Traceability Report DTP Workflow. The Traceability Report supports widgets that help you understand how well requirements are being implemented and tested.

Other Updates and Enhancements

DTP

- View both static analysis violations and suppressions in dashboard widgets and in the Violations Explorer
- Support for new Metric: METRIC.DIF - Nested If Statements
- Enhanced RuleMap support within the DTP Test Configuration UI
- v1.2 of the DTP REST API for violations, rules, files
- Support for Oracle 12c and MySQL 5.7
- Ships with JRE 8 and Tomcat 8
- The Source Control attributes of DTP Filters have been deprecated. Existing DTP Filters with Source Control settings will continue to use those settings, but new filters will no longer show options for configuring Source Control settings. There is no impact to the source control settings of the DTP Engines.

DTP Engines

- C/C++test: New rules for MISRA C:2012 (see New and Updated Code Analysis Rules.
- C/C++test: Enhanced support for Modern C++ standards (C++11, C++14, C++17), including dedicated static analysis. rules and test configuration.
- C/C++test: Added support for execution and coverage analysis of the CppUTest unit testing framework.
- Improved rules parameterization in the DTP Test Configuration UI.
- Improved reporting of errors and setup problems within HTML reports and on DTP.

New and Updated Code Analysis Rules

Rule ID	Description	Engine
CODSTA-MCPP-01	User-conversion cast operators should be made explicit	C/C++test
CODSTA-MCPP-02	Prefer alias declarations to typedefs	C/C++test
CODSTA-MCPP-03	Prefer Scoped Enums to Unscoped Enums	C/C++test
CODSTA-MCPP-04	Prefer 'nullptr' over 'NULL' or '0'(zero)	C/C++test
CODSTA-MCPP-05	Declare overriding functions with 'override' specifier	C/C++test
CODSTA-MCPP-06_a	Declare copy constructor and copy assignment operators with the 'delete' specifier to prevent copying of class	C/C++test
CODSTA-MCPP-06_b	Declare copy constructor and copy assignment operators with the 'delete' specifier instead of using a base class with private methods to prevent copying of class Coding Conventions	C/C++test
CODSTA-174_a_c90	A program should not exceed the translation limits imposed by The Standard (c90)	C/C++test
CODSTA-174_a_c99	A program should not exceed the translation limits imposed by The Standard (c99)	C/C++test
CODSTA-174_b_c90	A program should not exceed the translation limits imposed by The Standard (c90)	C/C++test
CODSTA-174_b_c99	A program should not exceed the translation limits imposed by The Standard (c99)	C/C++test
CODSTA-175_a	A function should not contain unused type declarations	C/C++test
CODSTA-175_b	A source file should not contain unused type declarations	C/C++test
CODSTA-176_a	A function should not contain unused local tag declarations	C/C++test
CODSTA-176_b	A source file should not contain unused tag declarations	C/C++test

CODSTA-177	A source file should not contain unused macro declarations	C/C++test
CODSTA-178	External identifiers shall be distinct	C/C++test
CODSTA-179_a_c90	Identifiers declared in the file scope and in the same name space shall be distinct (c90)	C/C++test
CODSTA-179_a_c99	Identifiers declared in the file scope and in the same name space shall be distinct (c99)	C/C++test
CODSTA-179_b_c90	Identifiers declared in the same block scope and name space shall be distinct (c90)	C/C++test
CODSTA-179_b_c99	Identifiers declared in the same block scope and name space shall be distinct (c99)	C/C++test
CODSTA-180	Identifiers that define objects or functions with external linkage shall be unique	C/C++test
CODSTA-181	The +, -, += and -= operators should not be applied to an expression of pointer type	C/C++test
CODSTA-182	The 'sizeof' operator shall not have an operand which is a function parameter declared as "array of type"	C/C++test
CODSTA-183	The pointer arguments to the Standard Library functions 'memcmp', 'memmove' and 'memcmp' shall be pointers to qualified or unqualified versions of compatible types	C/C++test
CODSTA-184	The pointer arguments to the Standard Library function 'memcmp' shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type	C/C++test
CODSTA-185_a	The pointers returned by the Standard Library functions 'localeconv', 'getenv', 'setlocale' or, 'strerror' shall only be used as if they have pointer to const-qualified type	C/C++test
CODSTA-185_b	Strings pointed by members of the structure 'lconv' should not be modified	C/C++test
CODSTA-186	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	C/C++test
BD-API-BADPARAM	Do not pass incorrect values to library functions	C/C++test
BD-API-CTYPE	Do not pass incorrect values to ctype.h library functions	C/C++test
BD-API-STRSIZE	The size_t argument passed to any function in string.h shall have an appropriate value	C/C++test
BD-API-VALPARAM	Validate values passed to library functions	C/C++test
BD-PB-EOFCOMP	The macro EOF should be compared with the unmodified return value from the Standard Library function	C/C++test
BD-PB-ERRNO	Properly use errno value	C/C++test
BD-PB-INVRET	Pointers returned by certain Standard Library functions should not be used following a subsequent call to the same or related function	C/C++test

BD-PB-MCCSTR	The Standard Library function memcmp shall not be used to compare null terminated strings	C/C++test
BD-PB-NORETURN	Never return from the function with 'noreturn' attribute	C/C++test
BD-PB-WRRDSTR	The same file shall not be opened for read and write access at the same time on different streams	C/C++test
BD-SECURITY-TDCONSOLE	Avoid printing tainted data on the output console	C/C++test
FORMAT-25_b	Parenthesis shall be used with the "return" statement	C/C++test
MISRA2004-9_2_b	Arrays shall not be partially initialized	C/C++test
MISRA2004-9_2_c	The non-zero initialization of structures requires an explicit initializer for each element	C/C++test
MISRA2012-DIR-4_5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	C/C++test
MISRA2012-DIR-4_7_a[b]	Consistently check the returned value of non-void functions	C/C++test
MISRA2012-DIR-4_7_b[b]	Always check the returned value of non-void function	C/C++test
MISRA2012-DIR-4_11[b]	Validate values passed to library functions	C/C++test
MISRA2012-DIR-4_13_a[b]	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	C/C++test
MISRA2012-DIR-4_13_b[b]	Do not use resources that have been freed	C/C++test
MISRA2012-DIR-4_13_c[b]	Do not free resources using invalid pointers	C/C++test
MISRA2012-DIR-4_13_d[b]	Do not abandon unreleased locks	C/C++test
MISRA2012-DIR-4_13_e[b]	Avoid double locking	C/C++test
MISRA2012-DIR-4_14_a[b]	Avoid tainted data in array indexes	C/C++test
MISRA2012-DIR-4_14_b[b]	Protect against integer overflow/underflow from tainted data	C/C++test
MISRA2012-DIR-4_14_c[b]	Avoid buffer read overflow from tainted data	C/C++test
MISRA2012-DIR-4_14_d[b]	Avoid buffer write overflow from tainted data	C/C++test
MISRA2012-DIR-4_14_e[b]	Protect against command injection	C/C++test
MISRA2012-DIR-4_14_f[b]	Protect against file name injection	C/C++test
MISRA2012-DIR-4_14_g[b]	Protect against SQL injection	C/C++test
MISRA2012-DIR-4_14_h[b]	Prevent buffer overflows from tainted data	C/C++test
MISRA2012-DIR-4_14_i[b]	Avoid buffer overflow from tainted data due to defining incorrect format limits	C/C++test
MISRA2012-DIR-4_14_j[b]	Protect against environment injection	C/C++test
MISRA2012-DIR-4_14_k[b]	Avoid printing tainted data on the output console	C/C++test

MISRA2012-RULE-1_1_a_c90	A program should not exceed the translation limits imposed by The Standard (c90)	C/C++test
MISRA2012-RULE-1_1_a_c99	A program should not exceed the translation limits imposed by The Standard (c99)	C/C++test
MISRA2012-RULE-1_1_b_c90	A program should not exceed the translation limits imposed by The Standard (c90)	C/C++test
MISRA2012-RULE-1_1_b_c99	A program should not exceed the translation limits imposed by The Standard (c99)	C/C++test
MISRA2012-RULE-1_3_a[b]	Avoid division by zero	C/C++test
MISRA2012-RULE-1_3_b[b]	Avoid use before initialization	C/C++test
MISRA2012-RULE-1_3_c[b]	Do not use resources that have been freed	C/C++test
MISRA2012-RULE-1_3_d	Avoid overflow when reading from a buffer	C/C++test
MISRA2012-RULE-1_3_e[b]	Avoid overflow when writing to a buffer	C/C++test
MISRA2012-RULE-1_3_f	The value of an expression shall be the same under any order of evaluation that the standard permits	C/C++test
MISRA2012-RULE-1_3_g	Don't write code that depends on the order of evaluation of function arguments	C/C++test
MISRA2012-RULE-1_3_h	Don't write code that depends on the order of evaluation of function designator and function arguments	C/C++test
MISRA2012-RULE-1_3_i	Don't write code that depends on the order of evaluation of expression that involves a function call	C/C++test
MISRA2012-RULE-1_3_j	Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression	C/C++test
MISRA2012-RULE-1_3_k	Do not use more than one volatile in one expression	C/C++test
MISRA2012-RULE-1_3_l	Don't write code that depends on the order of evaluation of function calls	C/C++test
MISRA2012-RULE-1_3_m	A function shall not return a pointer or reference to a non-static local object	C/C++test
MISRA2012-RULE-1_3_n	The address of an object with automatic storage shall not be assigned to an object which persists after the object has ceased to exist	C/C++test
MISRA2012-RULE-1_3_o	The left-hand operand of a right-shift operator shall not have a negative value	C/C++test
MISRA2012-RULE-2_3_a	A function should not contain unused type declarations	C/C++test
MISRA2012-RULE-2_3_b	A source file should not contain unused type declarations	C/C++test
MISRA2012-RULE-2_4_a	A function should not contain unused local tag declarations	C/C++test
MISRA2012-RULE-2_4_b	A source file should not contain unused tag declarations	C/C++test
MISRA2012-RULE-2_5	A source file should not contain unused macro declarations	C/C++test
MISRA2012-RULE-5_1	External identifiers shall be distinct	C/C++test

MISRA2012-RULE-5_2_a_c90	Identifiers declared in the file scope and in the same name space shall be distinct (c90)	C/C++test
MISRA2012-RULE-5_2_a_c99	Identifiers declared in the file scope and in the same name space shall be distinct (c99)	C/C++test
MISRA2012-RULE-5_2_b_c90	Identifiers declared in the same block scope and name space shall be distinct (c90)	C/C++test
MISRA2012-RULE-5_2_b_c99	Identifiers declared in the same block scope and name space shall be distinct (c99)	C/C++test
MISRA2012-RULE-5_8	Identifiers that define objects or functions with external linkage shall be unique	C/C++test
MISRA2012-RULE-8_6	An identifier with external linkage shall have exactly one external definition	C/C++test
MISRA2012-RULE-8_7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit	C/C++test
MISRA2012-RULE-9_4	An element of an object shall not be initialized more than once	C/C++test
MISRA2012-RULE-9_5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	C/C++test
MISRA2012-RULE-12_5	The 'sizeof' operator shall not have an operand which is a function parameter declared as "array of type"	C/C++test
MISRA2012-RULE-18_1_a[b]	Avoid accessing arrays out of bounds	C/C++test
MISRA2012-RULE-18_1_b[b]	Avoid accessing arrays and pointers out of bounds	C/C++test
MISRA2012-RULE-18_4	The +, -, += and -= operators should not be applied to an expression of pointer type	C/C++test
MISRA2012-RULE-20_8	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	C/C++test
MISRA2012-RULE-21_13[b]	Any value passed to a function in <ctype.h> shall be representable as an 'unsigned char' or be the value 'EOF'	C/C++test
MISRA2012-RULE-21_14[b]	The Standard Library function 'memcmp' shall not be used to compare null-terminated strings	C/C++test
MISRA2012-RULE-21_15	The pointer arguments to the Standard Library functions 'memcmp', 'memmove' and 'memcmp' shall be pointers to qualified or unqualified versions of compatible types	C/C++test
MISRA2012-RULE-21_16	The pointer arguments to the Standard Library function 'memcmp' shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type	C/C++test
MISRA2012-RULE-21_17_a[b]	Avoid overflow due to reading a not zero terminated string	C/C++test
MISRA2012-RULE-21_17_b[b]	Avoid overflow when writing to a buffer	C/C++test
MISRA2012-RULE-21_18[b]	The 'size_t' argument passed to any function in <string.h> shall have an appropriate value	C/C++test

MISRA2012-RULE-21_19_a	The pointers returned by the Standard Library functions 'localeconv', 'getenv', 'setlocale' or, 'strerror' shall only be used as if they have pointer to const-qualified type	C/C++test
MISRA2012-RULE-21_19_b	Strings pointed by members of the structure 'lconv' should not be modified	C/C++test
MISRA2012-RULE-21_20[b]	Pointers returned by certain Standard Library functions should not be used following a subsequent call to the same or related function	C/C++test
MISRA2012-RULE-22_3[b]	The same file shall not be opened for read and write access at the same time on different stream	C/C++test
MISRA2012-RULE-22_4[b]	Avoid writing to a stream which has been opened as read only	C/C++test
MISRA2012-RULE-22_7[b]	The macro 'EOF' should be compared with the unmodified return value from the Standard Library function	C/C++test
MISRA2012-RULE-22_8[b]	The value of 'errno' shall be set to zero prior to a call to an errno-setting-function	C/C++test
MISRA2012-RULE-22_9[bc]	The value of 'errno' shall be tested against zero after calling an errno-setting-function	C/C++test
MISRA2012-RULE-22_10[bc]	The value of 'errno' shall only be tested when the last function to be called was an errno-setting-function	C/C++test
MISRA2008-8_5_2_b	Arrays shall not be partially initialized	C/C++test
MISRA2008-8_5_2_c	Structures shall not be partially initialized	C/C++test
NAMING-50	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	C/C++test
PB-69	An element of an object shall not be initialized more than once	C/C++test
PREPROC-19	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	C/C++test
PB.INOE	Use String.IsNullOrEmpty to check if a string is null or empty	dotTEST

Resolved PRs/FRs

PR/FR ID	Description
117075	The Categories used by the "Compliance by Category" Widget don't match the target Test Configurations
119444	Unable to see Test Configuration in DTP Server after it is created
119478	Required feature for Server Edition license is not available with ENT licenses
120275	Jenkins Cobertura Coverage widget does not work with Japanese locale
118760	Rule UC.UP is missing in Jtest 10.x

118762	Rule METRIC.DIF is missing in Jtest 10.x
118761	Rule GLOBAL.ACD is missing in Jtest 10.x
120444	Gradle 3.0 not supported
118860	IntelliJ: sourcelevel in json file is empty
119814	Bulitin Test Configurations are not available in IBM RAD GUI
120140	Discrepancy between documentation and jtestcli.jvm regarding -Xmx value
120595	NAMING.NE does not work correctly in case of inner classes
121001	Wrong example repair code in rule's documentation of HIBERNATE.SLM
116773	BD-PB_NOTINIT stops triggering when unrelated code is added
116892	Add nullptr support in rulewizard
118116	Error: use of __if_exists is not supported in this context
119825	MISRA2008-5.0.6.a False Negative for expression initializers
120019	Differences between the results of the analysis when incremental mode is enabled or disabled
120121	Parsing error: a derived class is not allowed here
120126	Problem when -include/-exclude file is not found
120353	BD.PB.VOVR parameter name is incorrect and should be updated
120455	MISRA2004-9_2 false positive when #define symbol used in array initializer
120123	Incorrect use of website option
120203	Invalid MSBuild target when building WebSite with special characters in name or located in a solution subfolder
120296	Coverage markers are not visible when importing results to a project with path including Japanese characters
117336	Randomly failing Engine analysis
120151	Setup problems reported during diff operation on TFS
120315	Suppressions are ignored when rule id includes severity

1. Available for Eclipse-based IDEs.
2. Requires license for Flow Analysis; contact your Parasoft representative.
3. When using these rules in a custom test configuration the following parameters need to be set:
   For MISRA2012-RULE-22_8:
   ```
   MISRA2012-RULE-22_8-reportOnMissingErrnoCheck=false
   MISRA2012-RULE-22_8-reportOnUnnecessaryErrnoCheck=false
   ```

   For MISRA2012-RULE-22_9:
   ```
   MISRA2012-RULE-22_9-reportWhenErrnoIsNotZero=false
   MISRA2012-RULE-22_9-reportOnUnnecessaryErrnoCheck=false
   ```

   For MISRA2012-RULE-22_10:
   ```
   MISRA2012-RULE-22_10-reportWhenErrnoIsNotZero=false
   MISRA2012-RULE-22_10-reportOnMissingErrnoCheck=false
   ```

   These parameters are already set in the built-in "MISRA C 2012" test configuration