Parasoft Unit Testing Starter Kit

Unit Testing Best Practices: Tools for Unit Testing Success

Trane Case Study

Wipro Case Study

Parasoft Data Sheets

Unit Testing Best Practices:
Tools for Unit Testing Success

This paper is the culmination of interviews with Parasoft co-founder and former CEO Adam Kolawa. Mr. Kolawa discussed why, when, and how to apply essential software verification methods. The interviews also address code analysis, code review, memory error detection, message/protocol testing, functional testing, and load testing. In this interview, Mr. Kolawa counters claims that unit testing has peaked, explains how unit testing delivers value above and beyond "early testing," and warns that code coverage is not the best measure of test suite completeness.

Is unit testing more popular in theory than in practice?

Well, there's a huge barrier to entry because setting up realistic and useful tests can be quite complex. People tend to have a good understanding of how the application is supposed to work from the outside—how it's supposed to respond to different user actions or operations—but they have a limited grasp of how it's supposed to work "under the hood." This is quite a handicap when it comes to writing unit tests.

With unit tests, you need to create relevant initial conditions so the part of the application you're trying to test behaves like part of the complete system. If these initial conditions aren't set correctly, the test won't be exercising the code in a realistic context, and it won't be terribly useful to you. You can't really tell if something is working properly if you're not testing it under realistic conditions. The context is critical. For instance, a beam might provide proper support under many conditions--but if you're building with it, you really need to know if it will hold up under your specific construction. If it wasn't verified to work in a similar context, you can't rest assured that it will work for you here.

Creating the initial conditions can be difficult, but it's absolutely essential. I think the people who do try to accomplish this are frustrated by the amount of work required, and those who don't are disappointed that they're not getting valuable results.

Is unit testing nearing extinction?

I think that unit testing is more valuable now than ever. But I think we need a different flavor of unit testing than the one everyone always focuses on, which is early testing. Let me explain.

The systems we're building today, such as SOA and RIA, are growing in size and complexity—plus developers are being asked to evolve them faster than ever before. We typically have a good idea of what functionality we need to test because the use cases are usually well-defined in requirements. However, their complexity renders them difficult to test in their entirety: they are connected to the outside world, staging systems are difficult to establish and maintain, each test requires considerable setup, and so on. It might be feasible to test each use case once, but you really need to test all your use cases daily so that you're immediately alerted in case your constant modifications end up breaking them.

This is where unit testing comes in. Unit testing allows you to continuously test parts of the application without the hassles of dealing with the complicated system. Using a technology like Parasoft Tracer, you can create unit test cases that capture the functionality covered in your use cases. From the application GUI or using a SOA or Web test client, you just exercise your use cases while Tracer is enabled. As the use case is executed, Tracer monitors all the objects that are created, all the data that comes in and goes out, and then it creates you a unit test that represents this operation—and even sets up the proper initial conditions.

You can then take this unit test case and execute it on a separate machine, away from the original system. This means that you can use a single machine—even a standard developer desktop—to reproduce the behavior of a complicated system during your verification procedure. Add all of these tests to your regression test suite, run it continuously, and you'll immediately be alerted if this functionality that you captured is impacted by your code modifications.

Essentially, this helps you address the complexity of today's systems in two ways:

1. It relieves you from having to perform all the time-consuming and tedious work required to set up realistic test conditions.

2. It gives you a test case that you can run apart from the system—a test case you can run continuously and fully automatically.

Does creating unit tests after the application can realistically be exercised mesh with the idea that unit testing is early testing?

This type of unit testing is not early testing at all. It's a completely different flavor of unit testing, and I think it's even more valuable than early testing. But, there's still a time and place for early testing.

Say you have a brand new application that you're writing from scratch. As you implement each chunk of functionality, why not write a unit test that captures how this functionality is supposed to work? At this point, you're familiar with the requirement being implemented and you know all about how the code works—so this is the best possible time to set up the conditions properly and write a test that really verifies the requirement. While you're at it, specify what requirement it implements in the test case itself. This way, it can always be correlated to the appropriate requirement. In other words, if the test fails, you'll know what requirement was affected.

When you're done writing each test, add it to the regression test suite and execute it continuously—just like you would do with the other type of unit tests.

You could write these tests right after the code is implemented. Or you could write them before you even start working on code, essentially practicing TDD.

Is this requirement-based test development process always a manual one?

This process of writing tests that verify requirements is always a creative one. The developer really needs to think about the code and how to test it effectively. But, there is some help. Automated unit testing tools like Parasoft Development Testing Platform can reduce the work required to implement realistic and useful test cases.

For instance, the object repository stores initialized objects, which are very helpful to use when you're trying to set up realistic initial conditions. The stub library can be used to "stub out" external references so the unit can be tested in isolation. It's not a trivial task to set up stubs and objects manually, so having this assistance can save you a lot of time. Also, Test Case Parameterization can be used to feed additional inputs into the test cases—be it inputs from a data source, or a stream of automatically generated inputs using corner case conditions.

These things don't relieve you from having to think creatively about what the test should do, but they do help you implement your plan as efficiently as possible.

Do you agree with the widely-held belief that achieving greater code coverage with unit testing is more valuable?

Actually, I think that code coverage is irrelevant in most cases. People want high code coverage, but to achieve something like 80% code coverage or higher, you inevitably end up executing code mindlessly. It's like asking a pianist to cover 100% of the piano keys rather than hit just the keys that make sense in the context of a given piece of music. When he plays the piece, he gets whatever amount of key coverage makes sense.

Another problem is that when code coverage exceeds a certain level (above about 70%) the test suite becomes increasingly difficult to maintain. When the test suite has higher coverage than this, it's usually a sign that methods are executed mindlessly. You typically have very fine granularity assertions, and these assertions are very sensitive to code changes. As a result, many assertion failures are reported and need to be dealt with each day—but these assertion failures typically don't alert you to any significant changes.

Many people ask me if they can get 100% code coverage. The answer is yes—but it won't do you much good, other than satisfying some internal mandate for 100% code coverage.

How do you measure test suite completeness then?

While a piano concerto is complete when all the notes have been covered, the test suite is complete when every use case has been covered.

Each use case should have at least one test case. This might be a unit test, another test, or a combination of test types. When I have all of my use cases captured as test cases, then I can run my regression test suite nightly and it will tell me if the day's changes broke or negatively impacted the functionality related to my use cases. This gives you a safety net that's essential when you're constantly evolving the application.

Is there any point in even measuring code coverage then?

Yes, it actually tells you some interesting things. Say you have one test case for every use case, and this represents only 40% code coverage. This might mean that 60% of your code is unrelated to your use cases... which is a bit scary. Do you have a lot of useless code? Is the application being written without a clear understanding of what it's supposed to do? Or do you have major omissions in your use case definitions? Maybe you're missing some use cases for your requirements. Or maybe your requirements themselves are incomplete. This is definitely worth exploring.

On the other hand, what if your coverage is high even though you don't have many test cases? This might be a sign that you have well-written, tight code, with a lot of reuse.

Once you've built up a set of test cases that give you good coverage of your use cases, what do you do with them?

You need to ensure that they're maintained. Unit testing is not about creating unit test cases. It's about maintaining unit test cases. If you let them grow out of synch with the application, they quickly become useless.

To keep the process going, you need a supporting infrastructure and workflow. Ideally, each night the infrastructure automatically:

1. Gets the latest code from source control.
2. Runs the entire regression test suite.
3. Determines what assertions failed as a result of the day's modifications.
4. Figures out which developer caused each assertion failure.
5. Distributes this information to the responsible developers.

This is all automated, and doesn't require any development resources at all. Then, when the developers arrive at work each morning, they import the results into their desktops or IDEs, review any assertion failures reported for the code they authored, and respond to them. When they respond, they are either addressing functional defects in the code or updating the test to reflect the correct behavior of the code.

With this daily process, a little effort each morning goes a long way in terms of extending the test cases' value and life span—and also, of course, in terms of exposing unexpected impacts as they are introduced so you can nip them in the bud.

Without a supporting process and an infrastructure to drive this process and keep it on track, it's only a matter of time before your unit testing efforts decay.

How do you keep this daily process as painless as possible?

Well, the automated assertion failure assignment and distribution I just mentioned is key. We learned the value of this ourselves in the Parasoft development team. Years ago, when we started examining our unit testing process, we found that developers would write functional unit tests to verify requirements as code was implemented. However, when test assertions later failed as the application evolved, the failures weren't being addressed promptly.

In each case, someone needed to review the failure and decide if it was an intended change in behavior or an unwanted regression error. Our open source unit test execution tools couldn't tell us what tests were failing from whom, for what, since when. Instead, our nightly often produced a report that said that something like 150 of our 3,000+ tests failed. Each developer could not determine if his own tests failed unless he reviewed the full list of failures—one at a time—to see if it was something that he should fix or if it was a task for someone else. Our system was lacking accountability.

Initially, we tried asking all developers to review all test failures, but that took a lot of time and never became a regular habit. Then, we tried designating one person to review the test failures and distribute the work. However, that person was not fond of the job because it was tedious and the distribution of tasks was not well-received by the others.

Eventually, we built automated error assignment and distribution into our unit testing products, and started using it internally. Now, assertion failures are automatically assigned to the responsible developers based on source control data. If a developer causes one or more assertion failures, he is notified via email, imports only the relevant results into his IDE, and resolves them. As a result, failures are now resolved in days instead of months.

How do you begin and maintain a continuous process in an existing application lacking in test cases?

Working on a code base that has minimal tests or no tests is like walking on eggshells: every move you make has the potential to break something… but with software, the damage is not always immediately obvious. Before you touch another line of code, start building a unit test suite that serves as a change-detection safety net. This way, you can rest assured that you'll be alerted if modifications impact application behavior.

If you know what the use cases for the existing functionality are, you can start by "tracing" unit test cases as you execute the use cases. This way, you'll get a set of test cases that you can run daily to ensure that you're not breaking or changing this core functionality.

Even if you don't have use cases, you can automatically generate what I call a behavioral regression test suite: a baseline unit test suite that captures the code's current functionality. As I said before, test suite maintainability really diminishes above a certain code coverage level, so if your tool gives you a choice between generating a maintainable test suite and a high coverage one, choose the maintainable one. To detect changes from this baseline, ensure that the evolving code base is automatically run against this test suite on a regular basis—ideally, daily. You'll be alerted when the baseline functionality is impacted, and as you explore these impacts, you'll eventually learn more and more about the existing functionality.

Any closing comments?

Because today's applications are so complex, your quality efforts and regression test suites also need to be complex. Unit testing is a great starting point… but it's not a religion. Unit testing alone cannot deliver quality, will not expose all of your defects and change impacts, and is not a silver bullet.

If you really want to build quality into the code and gain a 360 degree view of how your daily modifications affect an existing application's functionality, you need a continuous quality process that includes everything from static analysis, to peer code reviews, to unit testing, to message/ protocol layer testing, to load testing. It takes some effort to get all the components established and working together, but the payoff in terms of team productivity as well as improved application quality is tremendous.

About Adam Kolawa

Adam Kolawa was truly a pioneer and a champion of software quality and developer productivity. He was considered an authority on the topic of software development and the leading innovator in promoting proven methods for enabling a continuous process for software quality. In 2007, eWeek recognized him as one of the 100 Most Influential People in IT.

Mr. Kolawa co-authored two books—*Automated Defect Prevention: Best Practices in Software Management* (Wiley-IEEE, 2007) and *Bulletproofing Web Applications* (Wiley, 2001), as well as contributed to O'Reilly's *Beautiful Code* book. He has also written or contributed to hundreds of commentary pieces and technical articles for publications such as *The Wall Street Journal*, *CIO*, *Computerworld*, and *Dr. Dobb's Journal*, as well as authored numerous scientific papers on physics and parallel processing.

Mr. Kolawa was granted 20 patents for software technologies he invented, including runtime memory error detection technology (Patent 5,842,019 and 5,581,696 - granted in 1998), statically analyzing source code quality using rules (Patent 5,860,011 - granted in 1999), and automated unit test case generation technology (Patent 5,761,408 and 5,784,553 - granted in 1998).

About Parasoft

For 25 years, Parasoft has researched and developed software solutions that help organizations define and deliver defect-free software efficiently. By integrating Development Testing, API/Cloud Testing, and Service Virtualization, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing with requirements traceability, functional & load testing, dev/test environment management, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently.

Contacting Parasoft

USA	Phone: (888) 305-0041	Email: info@parasoft.com
NORDICS	Phone: +31-70-3922000	Email: info@parasoft.nl
GERMANY	Phone: +49 731 880309-0	Email: info-de@parasoft.com
POLAND	Phone: +48 12 290 91 01	Email: info-pl@parasoft.com
UK	Phone: +44 (0)208 263 6005	Email: sales@parasoft-uk.com
FRANCE	Phone: (33 1) 64 89 26 00,	Email: sales@parasoft-fr.com
ITALY	Phone: (+39) 06 96 03 86 74	Email: c.soulat@parasoft-fr.com

OTHER See http://www.parasoft.com/contacts

Trane Smoothly Transitions to C++ Aided by Parasoft's Development Testing Platform

Trane (NYSE: TT) is a world leader in air conditioning systems, services, and solutions. They provide highly reliable and energy efficient comfort in commercial, industrial, institutional, and residential buildings throughout the world.

The Trane Global Modeling and Analysis team determined that moving to object-oriented development would ultimately enable more rapid and agile responses to business demands. By adopting C++ as the company's primary programming language, Trane could support a more component-based architecture for their code, which could be shared among numerous computer models. To help achieve this goal, Trane implemented Parasoft's Development Testing Platform.

Migrating from a Legacy System to C++

The Global Modeling and Analysis team is spread across three different locations: La Crosse, WI; Chicago, IL; and Shanghai, China. Some of these engineers work on code using C++ and VB.NET. Others work on mathematical models. The La Crosse group is responsible for the mathematical models.

The code for Trane's mathematical models is based on engineering rules, which changes very little over time. As a result, the legacy system had remained highly reliable for an extended period. Engineers wrote equations over the course of many years and the derived forms are relatively stable.

Even so, the advantages of moving to object-oriented development outweighed holding on to their legacy system. In addition to sharing components between their many computer models, the transition would enable the Global Modeling and Analysis team to integrate their in-house tools with multiple user interfaces—optimizing resources.

Finding a Quality Solution that Goes Above and Beyond Code Review

The Global Modeling and Analysis team created a list of coding standards to ensure that the code met uniform expectations around reliability, performance, and maintainability as they transitioned to C++ and .NET. To remain compliant with Six Sigma, Vikas Patnaik, Manager of Global Modeling and Analysis team, sought a process to validate the use and control of the new coding standards.

After researching code review methods, Jim Spielbauer, Trane Development Engineer, discovered that the manual code review processes were likely to impact the project schedule and budget, which led to the question: Is there any way to automate code reviews?

With a weighted list of features that focused mostly on verifying coding standards, Spielbauer and his teammate, Senior Software Developer Mike Eastey, started their search for automated testing software. They came across Parasoft's Development Testing Platform for C++ and .NET applications.

Spielbauer says, “Our list of desired features got bigger when we found Parasoft’s solution and realized its capabilities.” They were specifically drawn to automated unit testing, which allows them to start verifying the code’s reliability and functionality as each logical unit is completed. As a result, the length and cost of their downstream processes, such as debugging, are reduced.

Spielbauer states, “When we went into this, automated unit testing was something we didn’t realize we could get. Even though it wasn’t part of our criteria at the beginning, discovering we could get automated unit testing was a pleasant surprise.”

The platform’s integration with Visual Studio .NET, which enabled the engineers to test the code directly in their development environment, was another pleasant surprise. The engineers could develop code, then just click a button to test it with no additional project setup. Eastey remarks, “The fact that Parasoft’s solution can also integrate with Visual Studio .NET is a huge bonus.”

Spielbauer adds that, “It’s an important benefit that Parasoft solutions can work with both C++ and .NET languages. Parasoft provides a quality development solution that our entire team can grow with and experience continuous improvement.”

Transitioning with Ease

Deploying Parasoft’s Development Testing Platform eased the Global Modeling and Analysis team’s migration to C++ at a level that is both manageable and encouraging.

Because much of the team is new to C++, running their code through the platform’s C++ analysis component helps them learn best practices and techniques. Spielbauer states, “Since I am so new to C++, I do tend to make mistakes. Parasoft finds those errors early in the software development process. It enables me to fix the code before it reaches our users.”

He continues that Parasoft’s Development Testing Platform “is teaching us all to be better programmers. It helps us find errors that we didn’t even realize were errors.”

Spielbauer can create reusable tests and run them with nearly 100% coverage. The tests not only help him and his team expose structural errors as they are introduced, but also establish a regression test suite that determines if code modifications impact existing functionality.

Spielbauer explains, “Parasoft’s solution has saved the Global Modeling and Analysis team both time and resources that we would have otherwise spent finding and fixing defects. Instead, we get to spend that time adding new features and functionality.”

PARASOFT®

USA PARASOFT HEADQUARTERS
101 E. Huntington Drive, Monrovia, CA 91016
Phone: (888) 305-0041, Email: info@parasoft.com

Wipro Meets Exacting Software Quality Standards and Fuels Global Growth through Parasoft's Development Testing Platform

Wipro, a recognized provider of IT services to Global 1000 companies, has always emphasized the high quality of its code. The Bangalore-based company maintains software standards that often are far more rigorous than those that its clients have previously experienced or presently demand. Code testing coverage, for instance, must always reach 80% to meet Wipro's exacting standards.

Reaching these high levels of software quality in a rapid and cost-effective manner is challenging. Wipro's demanding objectives regarding code review and error reduction, led the company to implement Parasoft's automated Development Testing Platform. The move has certainly paid off; Wipro's software quality commitments have helped establish the firm as a leading force in the global IT services market, contributing to its dynamic growth and solid reputation for customer focused excellence.

Objective: Maintaining Exceptional Software Quality

Considering the whirlwind growth that Wipro has experienced in recent years, the challenge of maintaining high quality standards is always a top priority for the company. The constant initiation of new client projects means that code review is a persistent issue.

The firm's clients have stringent quality requirements, but Wipro's demands are often even more exacting. When the company conducts project evaluations, it rigorously tests software code to ensure it adheres to fixed quality standards.

However, rapid growth, intensifying competition, and complexities associated with mixed solutions eventually forced Wipro to find more efficient ways of meeting these standards. "We had to find ways of doing things faster, consistently and more dependably," says Vidya Kabra, Head of the Software Engineering Tools Group at Wipro.

Wipro needed an automated solution that could evaluate the entire code base against a single standard. Automated testing would be critical to ensure code reviewers would always deliver consistent and dependable reports without spending time on activities that could be handled by a tool.

Action: Implementing Parasoft Development Testing Platform

Wipro began exploring automated software testing solutions as a means for reconciling its commitment to software quality standards with its desire to continue driving growth. "Our challenges led us to a tool-based approach," explains Vidya. "It's not only software quality, but requirements, design, integration testing, regression testing, and unit testing that needed to be productively enhanced. We were under pressure to complete projects faster with quality built-in. The tool-based approach represented an automated approach--one that would save time and effort while meeting our quality goals."

After a rigorous review of potential solution providers and a series of pilots, Wipro chose to implement Parasoft's Development Testing Platform. "The products were reliable, customizable and cost effective," Vidya adds.

The Software Engineering Tools Group, which is responsible for procuring and advocating key software development tools, initially implemented Parasoft's development testing solution for Java, which enabled Wipro to automate and standardize code review. Parasoft's Development Testing Platform includes comprehensive code analysis for Java EE, SOA, Web, and other Java-based applications. "Parasoft has evolved well to become a comprehensive Java unit testing solution," says Sambuddha Deb, Chief Quality Officer, Wipro.

"Wipro has been using Parasoft successfully for years, and it is an excellent fit for Wipro's enterprise-wide Java development needs. By using Parasoft globally, we can deliver top-quality code to clients faster and more cost effectively."

Gradually, the group also integrated Parasoft's C, C++, and .NET development testing solutions, which enable businesses to automate and enforce their coding policies through static analysis, comprehensive code review, unit testing, and other practices.

Parasoft's Development Testing Platform enables teams to reduce the time and effort by 25% to reach code coverage objectives

Wipro's Software Engineering Tools Group is responsible for evangelizing the Parasoft Development Testing Platform across the company and across projects. "We showcase features of the products and pilot them," Vidya says. "We work with project teams and provide them with a scope-based usage approach for deploying the tools. Ultimately, the tools and new approaches get embedded in the organization. Our job is to set standards of quality within different project teams throughout Wipro and enable sustained commitment to these standards with code quality tools. Violation reports from these tools are also used as an input to code quality audits, which are run frequently by Wipro's audit office.

Results: Enhancing Code Review Productivity by 25%

Wipro's software quality standard requires projects to have 80% code coverage. Parasoft's development testing platform enables teams to reduce the time necessary to achieve the required code coverage by 25%, estimates Alexis Samuel, General Manager of Wipro's SEPG, Tools Group and Office of Productivity. "Despite the dramatic mix of size, technology and complexity of the projects that Wipro executes today, customer quality expectations are only increasing. Parasoft tools help us deliver a quality product commensurate with the technical depth that we are known for," he says.

Central to Wipro's success in the development process has been its policy of reviewing projects to determine how to drive continual improvement. For instance, a team working on a project in the manufacturing domain made extensive use of the powerful static analysis tools in Parasoft. The team was able to make the following code quality gains:

- Improve code coverage and ensured quality on 27.4 KLOC
- Identify and report 2060 violations; nearly all errors were fixed (230 minor violations were skipped) Automatically generate 1191 test cases, which contributed to overall code coverage.
- Meet customer requirements on code coverage with 23.84% effort savings.

In a separate case that revolved around static analysis, a Wipro team developed a printer driver using Parasoft's development testing solution for C and C++. The objective was to identify coding standards deviations in the development code. Parasoft enabled the team to identify 22,000 coding standards violations against 187 automated coding guidelines. Wipro's team met the customer's objectives in one third of the time that would have been required using manual resources for a coding standards adherence review.

Parasoft tools help us deliver a quality product commensurate
with the technical depth that we are known for.

Wipro has strengthened its position as a provider of high quality software through its usage of Parasoft's Development Testing Platform. "We have automated and standardized our best practices for providing customers the highest quality code," concludes Vidya. "We have dramatically improved the productivity of our testing efforts and this helps strengthen our position as a global provider of IT solutions."

PARASOFT®

USA PARASOFT HEADQUARTERS
101 E. Huntington Drive, Monrovia, CA 91016
Phone: (888) 305-0041, Email: info@parasoft.com

Parasoft Development Testing Platform

Parasoft's Development Testing Platform ensures the consistent application of software quality and security activities in the context of business expectations. For the first time, organizations can deploy a unified software development process that offers control over what needs to be developed as well as how it should be developed.

The Development Testing Platform automates static analsysis, unit testing, peer code review, coverage analysis, and runtime error detection, as well as accurately and objectively measures productivity and application quality.

Other benefits:

- Introduces quality practices as a continuous process, which is essential to agile or highly-iterative development.
- Provides unprecedented, real-time visibility into how the software is being developed and if it is meeting expectations.
- Reduces costs and risks throughout the entire SDLC by initiating a preventive strategy that helps developers detect and remediate defects before they become bugs.

Event- and Exception-Based Triggers

Automatically enforce policies during workflow-based events (e.g. code check-in) and exception-based events (e.g. static code analysis).

Infrastructure Integration

Simple integration with existing development infrastructure, including source control management, bug tracking, requirements management, and IDE.

Policy Management

- Define expectations about how the code should be written in understandable human language.
- Automatically enforce policies.
- Train software engineers on the business objectives driving policies.

Quality Practices

Execute a range of software analysis and tests, including:

- Static Code Analysis:
  - Integration-Time Analysis
  - Continuous-Integration Analysis
  - Edit-Time Analysis
  - Runtime Analysis
- Unit Testing
- Code Coverage Analysis
- Runtime Error Detection
- Peer Code Review
- API Testing
- Memory Error Detection
- Functional Testing
- Load Testing

Extensibility

Extend the Development Testing Platform through APIs, third-party plug-ins, and community-developed libraries to build a comprehensive development management ecosystem and enable cross-functionality within specialized environments.

Process Analysis Engine

- View metrics that focus on areas for improving the code.
- Gain early understanding of potential risks.
- Measure performance over time.
- Accurately quantify resource requirements and predict iteration durations.

Supported Languages and Technologies

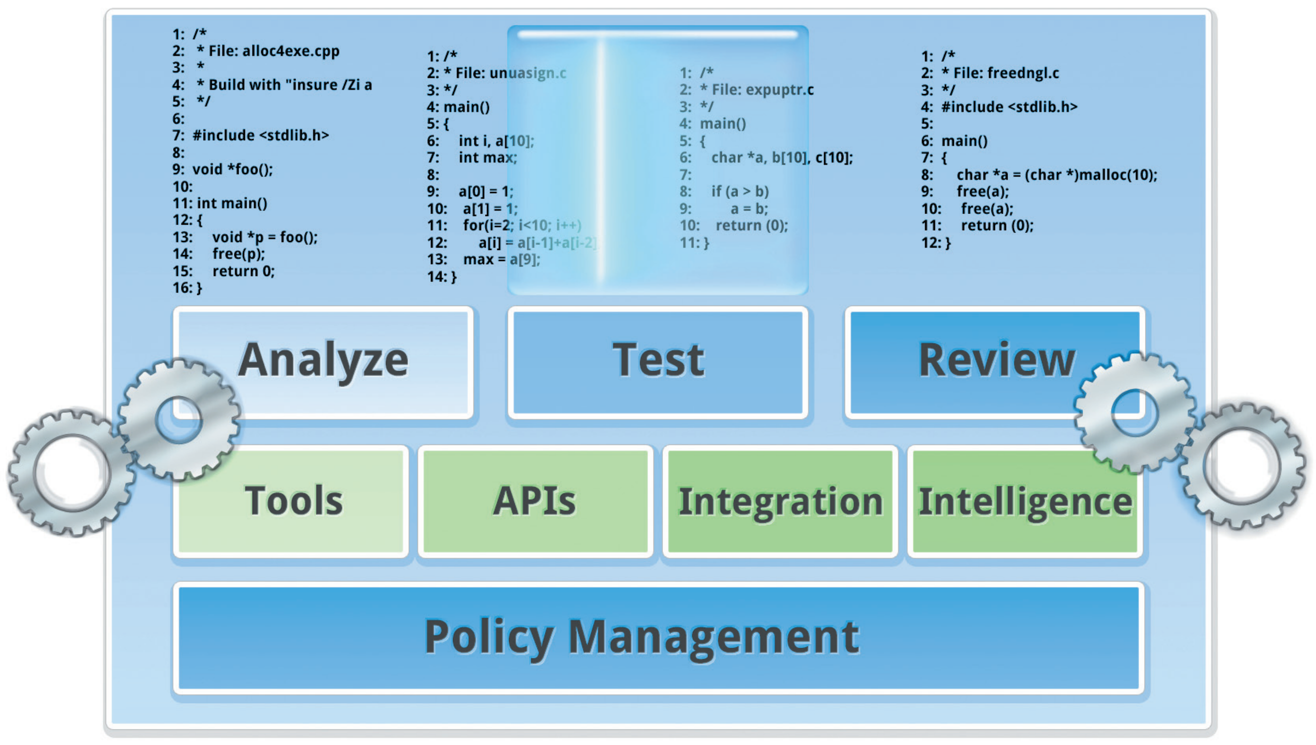
Java / JSP / XML / Android / Spring

Hibernate / Eclipse / JSF / Struts / JDBC

EJB / Servlets / .NET / C, C#, C++ / Managed C

VB.NET / ASP / Qt / STL

Parasoft Development Testing Platform



```
/*
 * File: alloc4exe.cpp
 *
 * Build with "insure /Zi a
 */

#include <stdlib.h>

void *foo();

int main()
{
    void *p = foo();
    free(p);
    return 0;
}
```

```
/*
* File: unuasign.c
*/
main()
{
    int i, a[10];
    int max;

    a[0] = 1;
     a[1] = 1;
     for(i=2; i<10; i++)
         a[i] = a[i-1]+a[i-2]
     max = a[9];
}
```

```
/*
* File: expuptr.c
*/
main()
{
   char *a, b[10], c[10];

   if (a > b)
       a = b;
    return (0);
}
```

```
/*
* File: freedngl.c
*/
#include <stdlib.h>

main()
{
    char *a = (char *)malloc(10);
    free(a);
     free(a);
     return (0);
}
```

PARASOFT®

USA PARASOFT HEADQUARTERS / 101 E. Huntington Drive, Monrovia, CA 91016
Phone: (888) 305-0041 / Email: info@parasoft.com

Parasoft C/C++test™

Parasoft C/C++test is an integrated development testing solution for C and C++. It automates a broad range of software quality practices—including static code analysis, unit testing, code review, coverage analysis, runtime error detection and more. C/C++test enables organizations to reduce risks, cut costs, increase productivity, and achieve compliance with industry guidelines and standards. It can be used in both host-based and target-based code analysis and test flows, which is critical for embedded and cross-platform development.

Automate Code Analysis for Monitoring Compliance

A properly implemented policy-driven development strategy can eliminate entire classes of programming errors by preventing defects from entering the code. C/C++test enforces your policy by analyzing code and reporting errors directly in the developer's IDE when code deviates from the standards prescribed in your programming policy.

Hundreds of built-in rules—including implementations of MISRA, MISRA C++, FDA, Scott Meyers' Effective C++, Effective STL, and other established sources—help identify bugs, highlight undefined or unspecified C/C++ language usage, enforce best practices, and improve code maintainability and reusability. Development managers can use the built-in rules and configurations or create highly specialized rules and configurations specific to their group or organization. Custom rules can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.

For highly quality-sensitive industries, such as avionics, medical, automobile, transportation, and industrial automation, C/C++test enables efficient and auditable quality processes with complete visibility into compliance efforts.

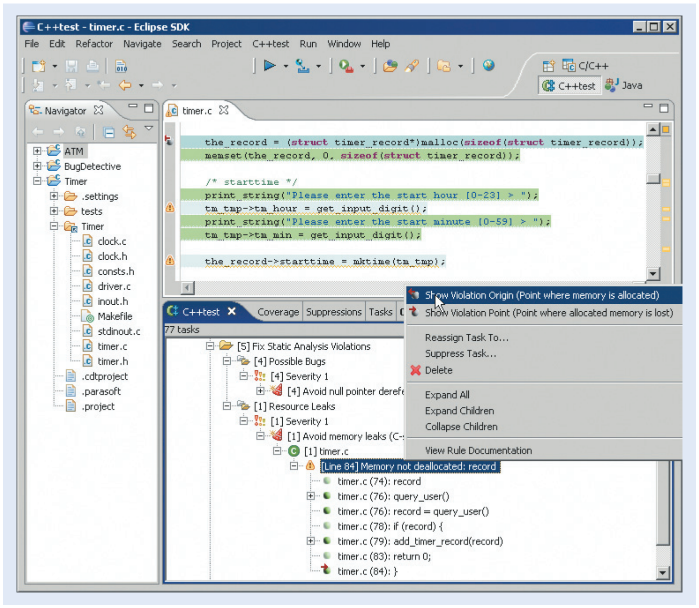
Identify Runtime Errors without Executing Software

C/C++test's integration-time static analysis module simulates feasible application execution paths—which may cross multiple functions and files—and determines whether these paths could trigger specific categories of runtime errors. The defects C/C++test detects include:

- Using uninitialized or invalid memory
- Null pointer dereferencing
- Array and buffer overflows
- Division by zero
- Memory and resource leaks
- Various flavors of dead code

The ability to expose defects without executing code is especially valuable for embedded applications, where detailed runtime analysis for such errors is often ineffective or impossible.

C/C++test greatly simplifies defect analysis by providing a complete highlighted path for each potential defect in the developer's IDE. Automatic cross-links to code help users quickly jump to any point in the highlighted analysis path.



```
the_record = (struct timer_record*)malloc(sizeof(struct timer_record));
memset(the_record, 0, sizeof(struct timer_record));

/* starttime */
print_string("Please enter the start hour [0-23] > ");
tm_tmp->tm_hour = get_input_digit();
print_string("Please enter the start minute [0-59] > ");
tm_tmp->tm_min = get_input_digit();

the_record->starttime = mktime(tm_tmp);
```

```
77 tasks
[5] Fix Static Analysis Violations
  [4] Possible Bugs
    [4] Severity 1
      [4] Avoid null pointer derefe
  [1] Resource Leaks
    [1] Severity 1
      [1] Avoid memory leaks (C-
        [1] timer.c
          [Line 84] Memory not deallocated: record
            timer.c (74): record
            timer.c (76): query_user()
            timer.c (76): record = query_user()
            timer.c (78): if (record) {
            timer.c (79): add_timer_record(record)
            timer.c (83): return 0;
            timer.c (84): }
```

C/C++test's customizable workflow allow users to test code as it's developed, then use the same tests to validate functionality/reliability in target environments.

Streamline Code Review

C/C++test automates preparation, notification, and tracking of peer code reviews, which enables an efficient team-oriented process. Status of all code reviews, including all comments by reviewers, is maintained and automatically distributed. C/C++test sup-ports two typical code review flows:

- **Post-commit code review**—Automatic identification of code changes in a source repository via custom source control interfaces; creates code review tasks based on pre-set mapping of changed code to reviewers.
- **Pre-commit code review**—Users can initiate a code review from the desktop by se-lecting a set of files to distribute or automatically identify all locally changed source code.

Additionally, the need for line-by-line inspections is virtually eliminated because the team's coding policy is monitored automatically with C/C++test's static analysis capability. By the time code is submitted for review, violations have already been identified and cleaned. Reviews can then focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect.

Unit and Integration Test with Coverage Analysis

C/C++test automatically generates complete tests, including test drivers and test cases for individual functions, purely in C or C++ code in a format similar to CppUnit. Auto-generated tests, with or without modifications, are used for initial validation of the func-tional behavior of the code. By using corner case conditions, the test cases also check function responses to unexpected inputs, exposing potential reliability problems.

Specific GUI widgets simplify test creation and management and a graphical Test Case Wizard enables developers to rapidly create black-box functional tests for selected functions without having to worry about their inner workings or embedded data dependencies. A Data Source Wizard helps parameterize test cases and stubs—enabling increased test scope and coverage with minimal effort. Stub analysis and generation is facilitated by the Stub View, which presents all functions used in the code and allows users to create stubs for any functions not available in the test scope—or to alter existing functions for specific test purposes. Test execution and analysis are centralized in the Test Case Explorer, which consolidates all existing project tests and provides a clear pass/fail status. These capabilities are especially helpful for supporting automated continuous integration and testing as well as "test as you go" development.

A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge the efficacy and completeness of the tests, as well as demonstrate compliance with test and validation requirements, such as DO-178B/C. Test coverage is presented via code highlighting for all supported coverage metrics—in the GUI or color-coded code listing reports. Summary coverage reports including file, class, and function data can be produced in a variety of formats.

Automated Regression Testing

C/C++test facilitates the development of robust regression test suites that detect if incremental code changes break existing functionality. Whether teams have a large legacy code base, a small piece of just-completed code, or something in between, C/C++test can generate tests that capture the existing software behavior via test assertions produced by automatically recording the runtime test results.

As the code base evolves, C/C++test reruns these tests and compares the current results with those from the originally captured "golden set." It can easily be configured to use different execution settings, test cases, and stubs to support testing in different contexts (e.g., different continuous integration phases, testing incomplete systems, or testing specific parts of complete systems). This type of regression testing is especially critical for supporting agile development and short release cycles, and ensures the continued functionality of constantly evolving and difficult-to-test applications.

Monitor and Eliminate Runtime Memory Errors

Runtime error detection constantly monitors for certain classes of problems—such as memory leaks, null pointers, uninitialized memory, and buffer overflows—and makes results available immediately after the test session is finished. The reported problems are presented in the developer's IDE along with details about how to fix the errors (including memory block size, array index, allocation/deallocation stack trace etc.). This not only improves the quality of the application—it also increases the skill level of your development staff.

Coverage metrics are collected during application execution. These can be used to see what part of the application was tested and to fine tune the set of regression unit tests (complementary to functional testing).

Test on the Host, Simulator, and Target

C/C++test automates the complete test execution flow, including test case generation, cross-compilation, deployment, execution, and loading results (including coverage metrics) back into the GUI. Testing can be driven interactively from the GUI or from the command line for automated test execution, as well as batch regression testing. In the interactive mode, users can run tests individually or in selected groups for easy debugging or validation. For batch execution, tests can be grouped based either on the user code they are liked with, or their name or location on disk.

C/C++test allows full customization of its test execution sequence. In addition to using the built-in test automation, users can incorporate custom test scripts and shell commands to fit the tool into their specific build and test environment. C++test's customizable workflow allows users to test code as it's developed, then use the same tests to validate functionality/reliability in target environments preset tool options.

C/C++test can be used with a wide variety of embedded OS and architectures, by cross-compiling the provided runtime library for a desired target runtime environment. All test artifacts of C/C++test are source code, and therefore completely portable.

Benefits

- **Increase productivity—**Apply a comprehensive set of best practices that reduce testing time, testing effort, and the number of defects that reach QA.
- **Achieve more with existing development resources—**Automatically vet known coding issues so more time can be dedicated to tasks that require human intelligence.
- **Increase code quality—**Efficiently construct, continuously execute, and maintain a comprehensive regression test suite that detects whether updates break existing functionality.
- G**ain unprecedented visibility into the development process—**Access on-demand objective code assessments and track progress towards quality and sched-ule targets.
- **Reduce support costs—**Automate negative testing on a broad range of potential user paths to uncover problems that might otherwise surface only in "real-world" usage.

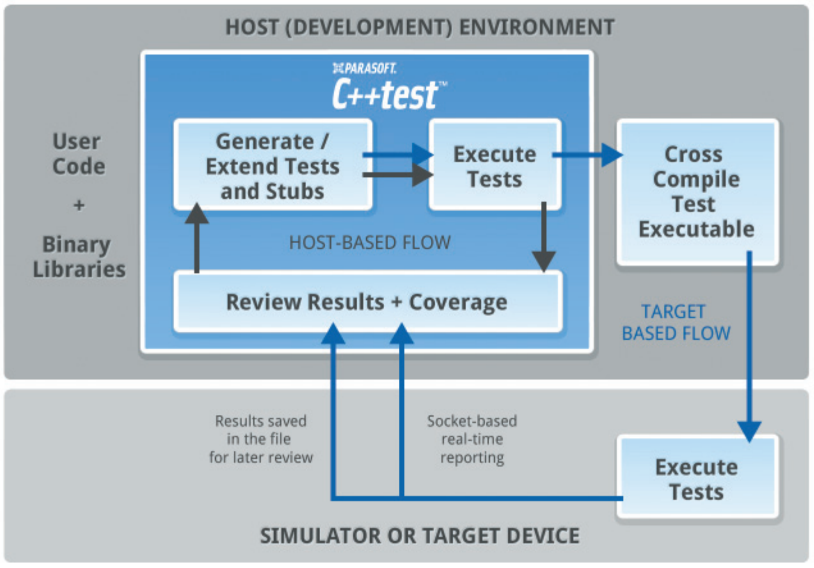
Key Features

- Static code analysis to reduce risks at each stage of development:
  - Integration-time analysis
  - Continuous integration-time analysis
  - Edit-time static analysis
  - Runtime static analysis
- Graphical rule editor for creating custom coding rules
- Automated generation and execution of unit and component-level tests
- Flexible stub framework
- Full support for regression testing
- Code coverage analysis with code highlighting
- Runtime memory error detection during unit test execution and application-level test-ing exposes hard-to-find errors, such as:
  - memory leaks
  - null pointers
  - uninitialized memory
  - buffer overflows
- Increase test result accuracy through execution of the monitored application in a real target environment
- HTML, PDF, and custom format reports:
  - Pass/fail summary of code analysis and test results
  - List of analyzed files
  - Code coverage summary
  - Reports can be automatically sent via email, based on a variety of role-based fil-ters
- Full team deployment infrastructure for desktop and command line usage

Supported Host Environments

Host Platforms
Windows / Linux / Solaris UltraSPARC

IDEs
- ARM Workbench / ARM Development Studio / ARM ADS
- Eclipse IDE for C/C++ Developers
- Green Hills MULTI
- IAR Embedded Workbench
- Keil µVision
- Microsoft eMbedded Visual C++ / Microsoft Visual Studio
- QNX Momentics IDE (QNX Software Development Platform)
- Texas Instruments Code Composer
- Wind River Tornado / Wind River Workbench


HOST (DEVELOPMENT) ENVIRONMENT
PARASOFT C++test
User Code + Binary Libraries
Generate / Extend Tests and Stubs
Execute Tests
Cross Compile Test Executable
HOST-BASED FLOW
Review Results + Coverage
TARGET BASED FLOW
Results saved in the file for later review
Socket-based real-time reporting
Execute Tests
SIMULATOR OR TARGET DEVICE

Host Compilers
Windows: Microsoft Visual Studio / GNU gcc/g++ / Green Hills MULTI
Linux 32 and 64 bit processor: GNU gcc/g++ / Green Hills MULTI
Solaris: Sun ONE Studio / GNU gcc/g++ / Green Hills MULTI

Target/Cross Compilers
Altera NIOS GCC / ADS (ARM Development Suite) / ARM for Keil uVision / ARM RVCT / ARM DS-5 GNU Compilation Tools / Cosmic Software 68HC08 / eCosCentric GCC / Freescale CodeWarrior C/C++ for HC12 / Fujitsu FR Family SOFTUNE / GCC (GNU Compiler Collection) / Green Hills MULTI / IAR C/C++ for ARM / IAR C/C++ for MSP430 / Keil C51 / Microsoft Visual C++ for Windows Mobile / Microsoft Embedded Visual C++ / QCC (QNX GCC) / Renesas SH SERIES C/C++ / STMicroelectronics ST20 / STMicroelectronics ST40 / TASKING 80C196 C / TASKING TriCore VX-toolset C/C++ / TI TMS320C2000 C/C++ / TI TMS320C54x C/C++ / TI TMS320C55x C/C++ / TI TMS320C6x C/C++ / TI TMS470 / TI MSP430 C/C++ / Wind River GCC / Wind River DIAB

Build Management
GNU make / Sun make / Microsoft nmake / ElectricAccelerator

Continuous Integration
Hudson / Jenkins / ElectricAccelerator

Source Control
AccuRev SCM / Borland StarTeam / CVS / Git / IBM Rational ClearCase / IBM Rational Synergy / Microsoft Team Foundation Server / Microsoft Visual SourceSafe / Perforce SCM / Serena Dimensions / Subversion (SVN)

PARASOFT®

USA PARASOFT HEADQUARTERS / 101 E. Huntington Drive, Monrovia, CA 91016
Phone: (888) 305-0041 / Email: info@parasoft.com

PARASOFT 25 YEARS

PARASOFT

Jtest®

DATA SHEET

Parasoft® Jtest® is an integrated solution for automating a broad range of practices proven to improve development team productivity and software quality. It focuses on practices for validating Java code and applications, and it seamlessly integrates with Parasoft SOAtest to enable end-to-end functional and load testing of today's complex, distributed applications and transactions.

Parasoft's customers, including the majority of the Fortune 500, rely on Jtest for:

- Preventing defects that impact application security, reliability, and performance
- Complying with internal or regulatory quality initiatives
- Ensuring consistency across large and distributed teams
- Increasing productivity by automating tedious yet critical defect-prevention practices
- Successfully implementing popular development methods like TDD, Agile, and XP

Capabilities

Static code analysis	Facilitates regulatory compliance (FDA, PCI, etc.). Ensures that the code meets uniform expectations around security, reliability, performance, and maintainability. Eliminates entire classes of programming errors by establishing preventive coding conventions.
Data flow static analysis	Detects complex runtime errors related to resource leaks, exceptions, SQL injections, and other security vulnerabilities without requiring test cases or application execution.
Metrics analysis	Identifies complex code, which is historically more error-prone and difficult to maintain.
Peer code review process automation	Automates and manages the peer code review workflow- including preparation, notification, and tracking- and reduces overhead by enabling remote code review on the desktop.
Unit test generation and execution	Enables the team to start verifying reliability and functionality before the complete system is ready, reducing the length and cost of downstream processes such as debugging.
Runtime error detection	Automatically exposes defects that occur as the application is exercised-including race conditions, exceptions, resource & memory leaks, and security attack vulnerabilities.
Test case "tracing"	Generates unit test cases that capture actual code behavior as an application is exercised providing a fast and easy way to create the realistic test cases required for functional/regression testing.
Automated regression testing	Generates and executes regression test cases to detect if incremental code changes break existing functionality or impact application behavior.
Coverage analysis	Assesses test suite efficacy and completeness using a multi-metric test coverage analyzer. This helps demonstrate compliance with test and validation requirements such as FDA.
Team deployment and workflow	Establishes a sustainable process that ensures software verification tasks are ingrained into the team's existing workflow and automated so team members can focus on tasks that truly require human intelligence.

These core capabilities are also available for C, C++, .NET languages.

Error assignment and distribution	Facilitates error review and correction. Each issue detected is prioritized, assigned to the developer who wrote the related code, and distributed to his or her IDE with direct links to the problematic code.
Centralized reporting	Ensures real-time visibility into quality status and processes. This helps managers assess and document trends, as well as determine if additional actions are needed for regulatory compliance.
Continuous "On-the-fly" static analysis	Automatically run static analysis in the background as developers review, add, and modify code. This helps the team identify and fix problems as soon as they are introduced.

Key Features

- Built-in support for Google Android, Spring, Hibernate, Eclipse plug-ins, TDD, JSF, Struts, JDBC, EJBs, JSPs, servlets, and more (mobile, embedded, Java EE...)
- Integrates with Parasoft SOAtest for end-to-end functional and load testing for web, SOA, and cloud development.
- Exposes runtime defects that occur as the application is exercised by unit, manual, or scripted tests–including race conditions, exceptions, resource leaks, and security attack vulnerabilities
- Without requiring execution, identifies execution paths that can trigger runtime defects
- Checks compliance to configurable sets of over 1000 built-in static analysis rules for Java
- Provides templates for OWASP Top 10, CWE-SANS Top 25, PCI DSS, and other security static standards
- Automatically corrects violations of 350+ rules with QuickFix
- Allows easy GUI-based customization of built-in rules
- Identifies and prevents concurrency defects such as deadlocks, race conditions, missed notification, infinite loops, data corruption other threading problems
- Automatically creates robust low-noise regression test suites–even for large code bases
- Generates functional JUnit test cases that capture actual code behavior as a deployed application is exercised
- Generates extendable JUnit and Cactus (in-container) tests that expose reliability problems and achieve high coverage using branch coverage analysis
- Integrates and extends manually-written unit test cases
- Continuously executes the test suite to identify regressions and unexpected side effects
- Performs runtime error detection as tests execute
- Parameterizes test cases for use with varied, controlled test input values (runtime-generated, user-defined, or from data sources)
- Monitors test coverage with multiple metrics
- Tracks code coverage from manual tests and test scripts
- Steps through tests with the debugger
- Tests individual methods, classes, or large, complex applications
- Calculates metrics such as Inheritance Depth, Lack Of Cohesion, Cyclomatic Complexity, Nested Blocks Depth, Number Of Children
- Identifies and refactors duplicate and unused code
- Automates the peer code review process (including preparations, notifications, and routing)
- Shares test settings and files team-wide or organization-wide
- Generates HTML, PDF, XML, and custom reports
- Tracks how test results and code quality change over time
- Provides GUI (interactive) and command-line (batch) mode

Infrastructure Support

- Full integration with Eclipse 3.2-3.7, IBM Rational Application Developer 7.0-8.0
- Integration with Ant, Maven, CruiseControl, Hudson, and other build & release tools
- Integration with most popular source control systems
- Open Source Control API, which allows teams to integrate any other source control system

System Requirements

Operating System
- Windows: 7, Vista, 2000, XP, or 2003 (x86 or x86_64)
- Linux: Red Hat E.L. 3, 4, 5 or equivalent (x86 or x86_64)
- Solaris: Solaris 10 (SPARC)
- Mac: OS X 10.5 or higher

Hardware
- Intel® Pentium® III 1.0 GHZ or higher recommended
- 512 MB RAM minimum; 2 GB RAM recommended
- JRE 1.3 or higher

www.parasoft.com

Contact info:
Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016
Ph: (888)305.0041, Fax: (626)256.6884, Email: info@parasoft.com

PARASOFT

Parasoft dotTEST™

DATA SHEET

Parasoft® dotTEST™ – Comprehensive Code Quality Tools for .NET Development

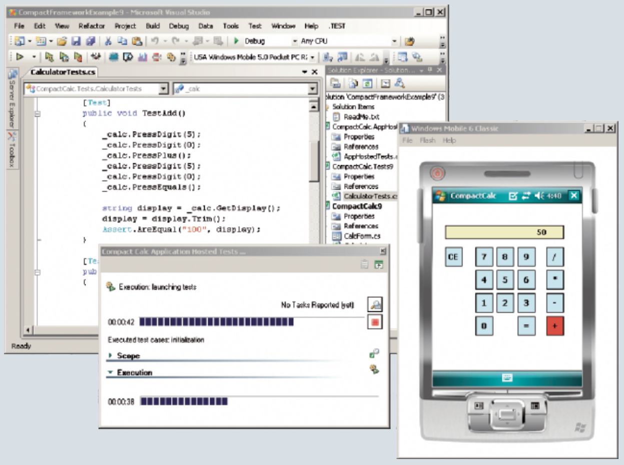
Parasoft® dotTEST™ is an integrated solution for automating a broad range of best practices proven to improve software development team productivity and software quality. dotTEST facilitates:

- Static analysis: Static code analysis, data flow static analysis, and metrics analysis
- Peer code review process automation: Preparation, notification, and tracking
- Unit testing: Unit test creation, execution, optimization, and maintenance
- Application testing: Sets up the actual application execution environment and launches tests from it

This provides teams a practical way to prevent, expose, and correct errors in order to ensure that their .NET code works as expected.

Tests can be run directly from Visual Studio or as part of an automated process. To promote rapid remediation, each problem detected is prioritized based on configurable severity assignments, automatically assigned to the developer who wrote the related code, and distributed to his or her IDE with direct links to the problematic code and a description of how to fix it.

Parasoft dotTEST works with programming languages that target the Microsoft .NET Framework and .NET Compact Framework, including C#, VB.NET, ASP.NET and Managed C++. It can test any file or assembly that has been built to take advantage of the .NET or .NET CF CLR.



```
[Test]
public void TestAdd()
{
    _calc.PressDigit(5);
    _calc.PressDigit(0);
    _calc.PressPlus();
    _calc.PressDigit(5);
    _calc.PressDigit(0);
    _calc.PressEquals();

    string display = _calc.GetDisplay();
    display = display.Trim();
    Assert.AreEqual("100", display);
}
```

```
Execution: launching tests
No Tasks Reported (yet)
00:00:42
Executed test cases: initialization
00:00:38
```

CompactFrameworkExample9 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Window Help .TEST
Debug
Any CPU
USA Windows Mobile 5.0 Pocket PC R:
CalculatorTests.cs
CompactCalc.Tests.CalculatorTests
_calc
Solution Explorer - Solution...
Solution 'CompactFrameworkExample9'
Solution Items
ReadMe.txt
CompactCalc.AppHost
Properties
References
AppHostedTests.
CompactCalc.Tests9
CalculatorTests.cs
CompactCalc9
CalcForm.cs
Ready
Compact Calc Application Hosted Tests ...
Scope
Execution
Windows Mobile 6 Classic
File Flash Help
CompactCalc
4:40
50
CE 7 8 9 / 4 5 6 * 1 2 3 - 0 = +

dotTEST's static analysis exposes critical defects early, without requiring code execution

Automate Code Analysis for Compliance

A properly-implemented coding policy can eliminate entire classes of programming errors by establishing preventive coding conventions. dotTEST statically analyzes code to check compliance with such a policy. To configure dotTEST to enforce a coding standards policy specific to their group or organization, teams can define their own rule sets with built-in and custom rules. dotTEST includes 400+ rules that cover Microsoft's .NET Framework Design Guidelines, CLS Compliance, Object Oriented Metrics, Security, and more.

In addition to rules that examine the IL code, dotTEST also provides rules that examine the C# source code; this enables dotTEST to check for many code issues that cannot be identified by IL-level analysis (for example, formatting issues, empty blocks, misuse of operators, etc.). Custom IL-level and C# rules, which are created with a graphical RuleWizard editor, can also enforce specific project and organizational requirements and prevent the recurrence of application-specific defects after a single instance has been found.

Benefits

Parasoft's customers, including 58% of the Fortune 500, rely on dotTEST for:

- Preventing defects that impact application security, reliability, and performance
- Complying with internal or regulatory quality initiatives Ensuring consistency across large and distributed teams
- Increasing productivity by automating tedious yet critical defect-prevention practices
- Successfully implementing popular development methods like TDD, Agile, and XP

DATA SHEET

DATA SHEET

Features

- Static analysis of code for compliance with user-selected coding standards
- Graphical RuleWizard editor for creating custom coding rules
- Static code path simulation for identifying potential runtime errors
- Streamlined code review process with a graphical interface and progress tracking
- Automated generation and execution of unit tests
- Generates functional unit test cases that capture actual code behavior as the application is exercised
- Launches tests from the actual execution environment
- Flexible stub framework for use in unit tests
- Full support for regression testing
- Code coverage analysis for unit testing and beyond (including application-level tests)
- Test directly on target devices or emulators
- Full team deployment infrastructure for desktop and command line usage
- Seamless integration with Microsoft Visual Studio

Platforms

- .NET Framework 2.0, 3.0, 3.5, 4
- .NET Compact Framework 2.0, 3.5
- Windows Mobile 5, Windows Mobile 6, Windows CE

System Requirements

- Windows 7, Windows Vista, Windows XP, Windows 2003 Server, Windows 2008 Server
- Visual Studio 2012, Visual Studio 2010, 2008 or Visual Studio 2005

Identify Runtime Bugs without Executing Software

BugDetective uses data flow analysis to detect runtime errors without requiring the software to actually be executed. This enables early and effortless detection of critical runtime errors that might otherwise take weeks to find. Defects detected include NullReferenceExceptions, ArgumentNullExceptions, resource leaks, division by zero, dereferencing before checking for null, SQL injections, XSS, and other security vulnerabilities.

Enable Effective and Comprehensive Team Code Review

The innovative Code Review module, which automates preparation, notification, and tracking of peer code reviews, addresses the known shortcomings of this very powerful development practice. dotTEST automatically identifies updated code, matches the code with designated reviewers, and tracks the progress of each review item until closure. With the Code Review module, teams can establish a bulletproof review process—where all new code gets reviewed and all identified issues are resolved.

Automate Unit and Component Testing for Instant Verification and Regression Testing

dotTEST's automated testing capabilities significantly reduce the work required to develop and maintain an effective test suite. dotTEST's automated testing capabilities are especially helpful for supporting continuous integration and agile/iterative development.

dotTEST provides numerous ground-breaking technologies to facilitate unit testing, including:

- **Unit Test Genie:** Allows you to generate specific object factory methods and test scenarios by interacting with dotTEST wizards. You can control precisely what objects and test scenarios are generated.
- **Non-Interactive Test Case Generation:** Allows you to create a large number of tests with minimal time and effort. This is especially useful for achieving high code coverage and establishing a regression baseline.
- **Application hosted testing:** Allows you to launch unit tests from virtually any point within your application—without changing your application or writing additional code. This lets you create complex objects in their natural environment and facilitates test development/maintenance.
- **Extensive coverage analysis:** Tracks coverage information for all tests—from dotTEST-based unit testing to manual application testing—and can combine the coverage information from multiple test runs. This helps you accurately gauge test suite efficacy and completeness, as well as demonstrate compliance with test and validation requirements.
- **Flexible stub support:** Allows classes to be tested in isolation. This addresses one of the greatest challenges in writing unit tests: getting complex objects in different states.

.NET Compact Framework Support

dotTEST's .NET Compact Framework support allows you to run unit tests directly on a device. This enables you to:

- Write very realistic unit tests because code runs against the .NET CF, which accurately represents realistic application behavior.
- Automatically check your code against any device or emulator that supports Windows Mobile Device Center (Active Sync) communication.
- Access an API, such as native API, which is available for a particular device only.

www.parasoft.com

Contact info:
Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016
Ph: (888) 305.0041, Fax: (626) 256.6884, Email: info@parasoft.com