PARASOFT®

TECHNICAL WHITEPAPER

Software Testing Methodologies Guide

WHAT ARE SOFTWARE TESTING METHODOLOGIES?

Software testing methodologies are the strategies, processes, or environments development teams use for testing. These methodologies enable developers, engineers, and programmers to conduct rigorous software testing as part of a go-to-market strategy. Common methodologies include:

- Waterfall
- Agile DevOps
- Iterative

The two most popular SDLC methodologies are Agile DevOps and waterfall, and testing is very different for these two environments. Following is a brief overview of all three methodologies.

WATERFALL

In the waterfall methodology, formal testing is conducted in the testing phase, which begins upon completion of the development phase. The waterfall methodology for testing works well for small, less complex projects. However, if requirements are not clearly defined at the start, it's extremely difficult to go back and make changes in completed phases.

The waterfall methodology is popular with small projects because it has fewer processes and players to tend with, which can lead to faster project completion. However, bugs are found later in development, making them more expensive to fix.

AGILE DEVOPS

The Agile methodology is best suited for larger development projects. Agile testing is an incremental methodology where testing is performed at the end of every increment or iteration.

Additionally, the whole application is tested upon completion of the project. There's less risk in the development process with the Agile methodology because each team member understands what has or has not been completed. The results of development projects are typically better with Agile when there's a strong, experienced project manager who can make quick decisions.

When taking a DevOps approach to testing, or continuous testing, there's a collaboration with operations teams through the entire product life cycle. Through this collaboration, development, and operations teams don't wait until the software is built or near completion to do testing. That means the software delivery process is faster, defects are detected earlier and are less expensive to resolve.

Continuous testing uses automated testing and automation tools as components of the software development pipeline to provide immediate feedback on any business risks that might exist.

ITERATIVE

In the iterative methodology, developers create basic versions of the software, review, and improve on the application in iterations—small steps. This is a good approach for extremely large applications that need to be completed quickly. Defects can be detected earlier, which means they can be less costly to resolve.

WHAT ARE THE TYPES OF SOFTWARE TESTING?

The most common types of software testing include:

- Static analysis
- Unit testing
- Integration testing
- System testing
- Acceptance testing

Other important types of testing not covered in this paper include:

- Fuzz testing
- Penetration testing
- Minimum & maximum value boundary testing
- Smoke testing
- Load testing
- Installation testing
- Interface testing

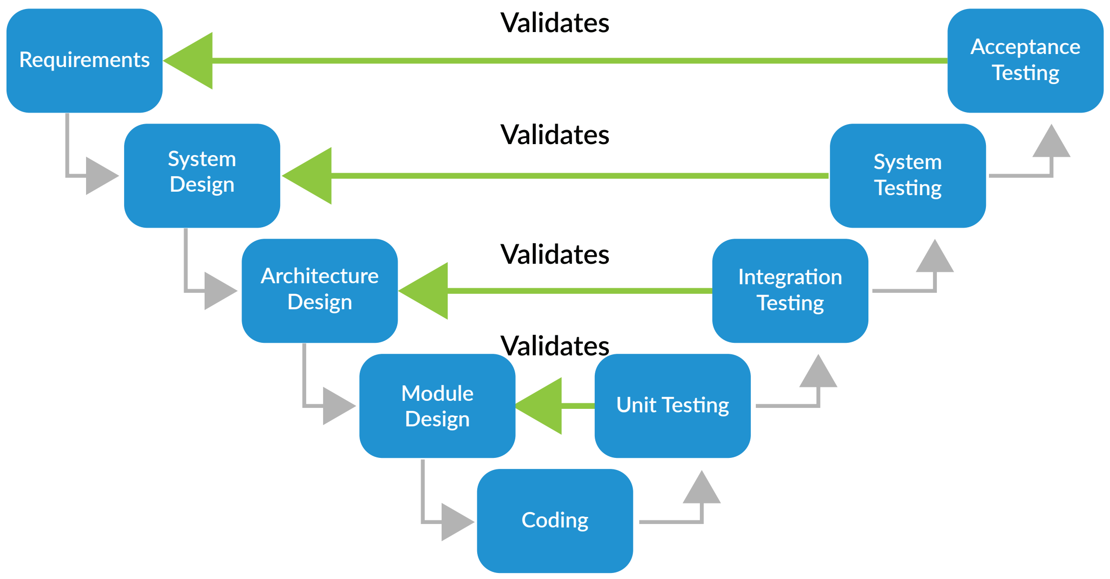


Figure 1:
V-model validation

STATIC ANALYSIS

Static analysis involves no dynamic execution of the software under test and can detect possible defects in an early stage before running the program. Static analysis is done during or after coding and before executing unit tests.

A code analysis engine can execute static analysis to automatically walk through the source code and detect noncomplying rules or lexical, syntactic, and even some semantic mistakes.

Static code analysis tools assess, compile, and check for vulnerabilities and security flaws to analyze code under test. Parasoft's static analysis tools help users manage the results of testing, including prioritizing findings, suppressing unwanted findings, and assigning findings to developers. These tools support a comprehensive set of development ecosystems to integrate into an extensive list of IDE products to conduct static analysis for C, C++, Java, C#, and VB.NET.

UNIT TESTING

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. Developers perform this type of testing before the setup is handed over to the testing team to formally execute the test cases.

Developers working on individual units of source code perform unit testing for their respective assigned areas. They use test data that's different from the test data of the quality assurance (QA) team.

Using Parasoft to perform branch, statement, and modified condition decision coverage (MC/DC) coverage is a form of unit testing. The software is isolated to each function and these individual parts are examined. The limitation of unit testing is that it cannot catch every bug in the application as it does not evaluate a thread or execution path in the application.

INTEGRATION TESTING

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways.

1. **Bottom-up integration testing.** Testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2. **Top-down integration testing.** In this testing, the highest level modules are tested first. Progressively, lower-level modules are tested thereafter.

SYSTEM TESTING

The QA testing team performs system testing on the entire system or application as a whole. The rigorous testing starts after all of the components are integrated and fully implemented. It's considered black box testing and there's no need to understand the inner workings of the system under test.

The system or application is tested thoroughly to verify that it meets functional requirements, quality of service requirements, and business requirements. QA performs the testing in the final production environment—or one that's very close—to where the application will be deployed.

When passing results are achieved, organizations gain a good sense of the time to market.

ACCEPTANCE TESTING

This is arguably the most important type of testing. The QA team conducts acceptance testing to gauge whether the application meets the intended specifications and satisfies the client's requirements. The QA team will have a set of pre-written scenarios and test cases to use to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are intended to point out simple spelling mistakes, cosmetic errors, and interface gaps, as well as to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

WHEN IS TESTING COMPLETED AND PUT TO AN END?

It's difficult to determine when to stop testing. Testing is a neverending process and no one can claim that software is 100% tested. However, there are criteria to consider that can serve as indicators for putting a stop to testing.

1. **Management decision.** The simplest and most common way to know testing is halted is when management decides to stop the testing process. Management's decision may be due to time or budget constraints, which may compromise quality. The decision may simply be that the project has reached the extent of required testing—meaning testing deadlines have been reached.
2. **Completion of test case execution.** Upon the completion of test cases, the test case pass rate should conform to product safety and security requirements. In some cases, for safety critical software, all test cases must pass.
3. **Completion of requirements and robustness testing.** Developers and testers can analyze data from the test results to make sure the application operates as expected and receives a passing result for every defined requirement. Additionally, all major functional flows are successfully executed with various inputs and work well.
4. **Code coverage to a pre-specified percentage.** Instrumenting your code and running all your test cases will provide a percentage of the code tested and expose code that has not been executed and potentially has hidden bugs. Some organizations are comfortable obtaining 80% and higher code coverage, while other organizations require 100% statement, branch, and MC/DC.
5. **The bug rate falls below a certain level.** When bug rates fall below a predetermined level and no high-priority bugs are identified, testing can be halted.

HOW DO AUTOMATED SOLUTIONS HELP WITH TESTING?

Your team can deliver quality software that's safe, secure, and reliable at scale with automated testing solutions that span every stage of the development life cycle. With solutions that provide a unified set of tools to accelerate testing, teams can shift testing left to the early stages of development while maintaining traceability, test result recordkeeping, code coverage details, report generation, and compliance documentation. Automated solutions provide the following types of testing:

- Static analysis
- Unit testing
- Integration testing
- System testing
- Acceptance testing
- Analysis and reporting
- Virtual test environment

STATIC ANALYSIS

Many of the quality tasks needed for software development, such as the need for a unit verification process and acceptance criteria for software units. Such acceptance criteria include data and control flow analysis, planned resource allocation, fault handling, memory management are supported by modern advanced tools like Parasoft C/C++test. In addition, static analysis tools include metrics and support peer code review with capabilities that assist unit testing and runtime error detection.

- **Control flow analysis.** A static code analysis technique for determining the control flow of a program. Modern advanced static analysis tools like Parasoft C/C++test use sophisticated control and data flow analysis to detect complex defects and security vulnerabilities.
- **Data flow analysis.** A technique for gathering information about the possible set of values calculated at various points in a computer program. Data flow analysis is a critical aspect of advanced static analysis tools that helps detect complex errors such as tainted data vulnerabilities.

Static analysis tools like Parasoft C/++test prevent and detect a variety of error and warning classes.

- **Initialization of variables.** Detects and warns developers about uninitialized variables.
- **Memory management and memory overflows.** Warns developers of various memory management bugs such as buffer overflows, memory leaks, resource leaks, and variable overflows (integer number overflows).
- **Boundary conditions.** Warns developers about poor type conversions where values might exceed type boundaries.

In addition, Parasoft C/C++ provides support for the following important development activities.

» **Walkthroughs, code reviews, and inspections.** Teams use these informal methods to verify design and implementation. Static analysis tools automate much of the tedious aspects of code inspection like coding standards compliance while flagging errors and possible software weaknesses.

» **Coding standard enforcement.** Static analysis tools are essential in establishing and enforcing a compliant coding standard such as MISRA. Although initially focused on automotive systems, MISRA is becoming a de facto standard in other industries. Another example that's suited for writing more secure code is SEI CERT C/C++, which is gaining traction across industries, too.

UNIT TESTING

In keeping with established industry quality standards, the following unit testing techniques can be satisfied and accelerated with test automation solutions.

» Automated test execution

» Automated test case generation

» Target-based testing for embedded devices

In simple terms, unit testing has two requirements.

1. Test cases that provide the context and data to drive the test.
2. An environment in which to run the test.

Test automation solutions like Parasoft C/C++test have tools that provide the test environment, such as a harness, mocked functions, and stubs. It also provides the data and management tools to execute and record the results of the tests, including code coverage and requirements traceability.

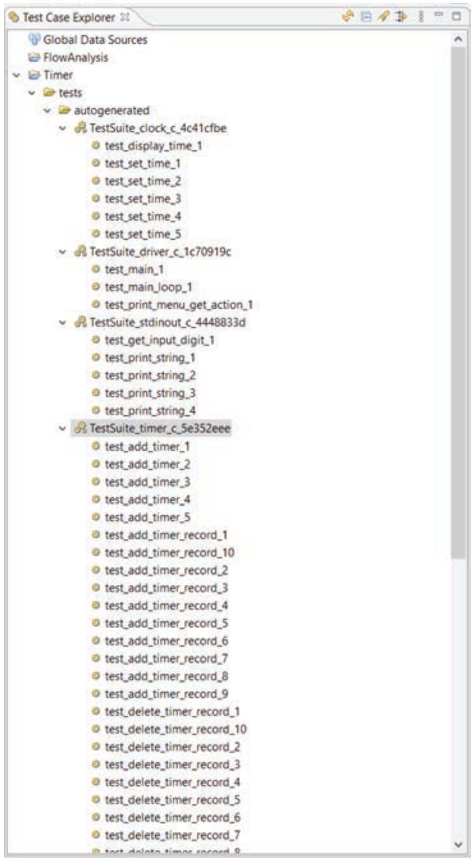
Automated Test Execution

Test automation provides large benefits to embedded software. Parasoft C/C++test automates test suites and reduces manual intervention so that testing can be done quicker, easier, and more often.

Offloading this manual testing effort frees up time for better test coverage and other safety and quality objectives. An important requirement for automated test suite execution is being able to run these tests on both host and target environments.

Unit test automation tools universally support some sort of test framework, which provides the harness infrastructure to execute units in isolation while satisfying dependencies via stubs. Parasoft C/C++test is no exception. Part of its unit test capability is the automated generation of test harnesses and the executable components needed for host and target-based testing.

Test data generation and management is by far the biggest challenge in unit testing. Test cases are particularly important in safety-critical software development because they must ensure functional requirements and test for unpredictable behavior, security, and safety requirements. All while satisfying test coverage criteria.



```
Test Case Explorer
  Global Data Sources
  FlowAnalysis
v Timer
  v tests
    v autogenerated
      v TestSuite_clock_c_4c41cfbe
          test_display_time_1
          test_set_time_1
          test_set_time_2
          test_set_time_3
          test_set_time_4
          test_set_time_5
      v TestSuite_driver_c_1c70919c
          test_main_1
          test_main_loop_1
          test_print_menu_get_action_1
      v TestSuite_stdinout_c_4448833d
          test_get_input_digit_1
          test_print_string_1
          test_print_string_2
          test_print_string_3
          test_print_string_4
      v TestSuite_timer_c_5e352eee
          test_add_timer_1
          test_add_timer_2
          test_add_timer_3
          test_add_timer_4
          test_add_timer_5
          test_add_timer_record_1
          test_add_timer_record_10
          test_add_timer_record_2
          test_add_timer_record_3
          test_add_timer_record_4
          test_add_timer_record_5
          test_add_timer_record_6
          test_add_timer_record_7
          test_add_timer_record_8
          test_add_timer_record_9
          test_delete_timer_record_1
          test_delete_timer_record_10
          test_delete_timer_record_2
          test_delete_timer_record_3
          test_delete_timer_record_4
          test_delete_timer_record_5
          test_delete_timer_record_6
          test_delete_timer_record_7
```

Figure 2:
Example of an automated test case generation. In this case, one test suite per function.

Automated Test Case Generation

Teams can automatically generate test cases like the popular CppUnit format with Parasoft C/C++test. By default, C/C++test generates one test suite per source/header file. It can also be configured to generate one test suite per function or one test suite per source file.

Safe stub definitions are automatically generated to replace "dangerous" functions, which include system I/O routines such as rmdir(), remove(), rename(), and so on. In addition, stubs can be automatically generated for missing function and variable definitions. User defined stubs can be added as needed.

Target-Based Testing for Embedded Devices

Automating testing for embedded software is more challenging due to the complexity of initiating and observing tests on embedded targets. Not to mention the limited access to target hardware that software teams have.

Software test automation is essential for making embedded testing workable on a continuous basis from host development system to target system. Testing embedded software is particularly time consuming. Automating the regression test suite provides considerable time and cost savings. In addition, test results and code coverage data collection from the target system are essential for validation and standards compliance.

Traceability between test cases, test results, source code, and requirements must be recorded and maintained. So, data collection is critical in test execution.

Parasoft C/C++test is offered with its test harness optimized to take minimal additional overhead for the binary footprint and provides it in the form of source code, where it can be customized if platform-specific modifications are required.

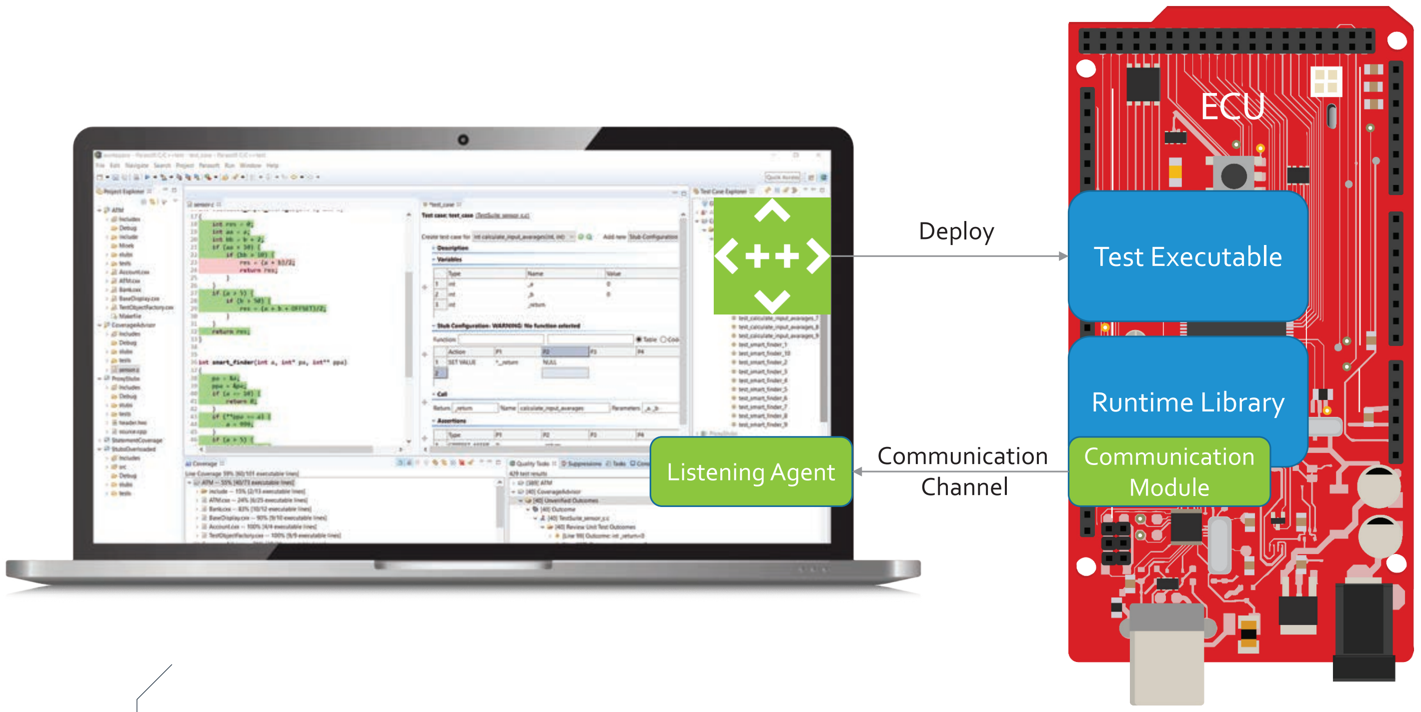


Figure 3: A high-level view of deploying, executing, and observing tests from host to target.

One huge benefit that the Parasoft C/C++test solution offers is its dedicated integrations with embedded IDEs and debuggers that make the process of executing test cases smooth and automated. Supported IDE environments include:

- Eclipse
- VS Code
- Green Hills Multi
- Wind River Workbench
- IAR EW
- ARM MDK
- ARM DS-5
- TI CCS
- Visual Studio
- Many others

INTEGRATION TESTING

Integration testing follows unit testing. The goal is to validate the architectural design by validating the behavior of multiple components together. Teams can perform integration testing from the bottom up and the top down with a combination of approaches.

Parasoft C/C++test automation supports integration testing by providing flexible test environments where stubs and mocks can be interchangeably enabled and disabled to test higher level functionality.

SYSTEM TESTING

System testing tests the system as a whole and focuses on validating requirements. Once all the components are integrated, the entire system is tested rigorously to verify that it meets the specified functional, safety, security, and other nonfunctional requirements.

Parasoft C/C++test can instrument the code to gather code coverage during system testing. This exposes code that hasn't been tested and where bugs may be lurking.

ACCEPTANCE TESTING

During system engineering, acceptance test cases are defined. The QA team performs them later in the SDLC. The acceptance test cases verify and validate whether the software meets the customer's requirements and expectations. With Parasoft C/C++test, users can realize the tests defined by the system engineers through test execution, then capture and provide proof showing the fulfillment of customer expectations.

If any acceptance tests fail, it means that customer expectations aren't being met, and contractual agreements are likely not being satisfied. The product should not be delivered until all acceptance tests have been validated.

ANALYSIS & REPORTING

Parasoft test automation solutions support the validation (actual testing activities) in terms of test automation and continuous testing. These solutions also support the verification of the activities, which means supporting the process and standards requirements. Key aspects of safety and security in embedded software development are requirements traceability and code coverage.

Many development standards require that software development plans include traceability between system requirements, software requirements, software system test, and risk control measures implemented in software. In addition, requirements analysis requires that software requirements are identified in such a way as to make it possible to demonstrate traceability between the requirement and software system testing.

Two Way Traceability

Requirements are the key driver for product design and development. These requirements include functional safety, security, and nonfunctional requirements that fully define the product.

This reliance on documented requirements is a mixed blessing because poor requirements are one of the critical causes of safety and security incidents in software. In other words, the implementation wasn't at fault, but poor or missing requirements were.

Automating Bidirectional Traceability

Maintaining traceability records on any sort of scale requires automation. Application life cycle management tools include requirements management capabilities that are mature and tend to be the hub for traceability.

Integrated software testing solutions like Parasoft complete the verification and validation of requirements by providing an automated bidirectional traceability to the executable test case. This includes the pass or fail result and traces down to the source code that implements the requirement.

Parasoft integrates with market leading requirements management tools or ALM systems.

- PTC Intland codebeamer
- Polarion from Siemens
- Atlassian Jira
- Jama Connect
- And more

Using any of Parasoft's test automation solutions within the development life cycle supports the association of tests with work items like requirements, defects, and test cases/test runs that are defined in these systems. Traceability is managed through Parasoft DTP's central reporting and analytics dashboard.

System Testing at the Service Level

Embedded systems often have connectivity into larger systems that collect and analyze patient data. For example, health care providers can track data and provide medical advice based on it. The ecosystem for medical devices is an important aspect of system testing since verification of data integrity, security, and confidentiality are required. System testing needs to include these environments to fully verify the new device.

Instead of viewing system quality in terms of meeting individual device requirements, the scope is broadened to consider the quality of the services provided. Testing at the service level ensures nonfunctional requirements are met. For example, performance and reliability are difficult to assess at the device level or during software unit testing. Service based testing can simulate the operational environment of a device to provide realistic loads.

Security is a significant concern in medical devices. Cyberattacks most likely originate from the network itself by attacking the exposed APIs. Service based testing can create simulated environments for robust security testing, either through fuzzing (random and erroneous data inputs) or denial-of-service attacks.

VIRTUAL TEST ENVIRONMENTS

To expedite software development and testing, teams can benefit from using virtual ecosystems like cloud SaaS deployments, virtual machines, and containerized environments. Teams can customize or create virtual environments or employ ready-to-use images containing everything they need to develop, build, run, and test their applications.

Companies, like Docker, Arm, VMware, Amazon, and more, provide virtual environments, which save physical space and reduce time to market and management costs.

In addition, because hardware may not be available or is expensive to purchase, many companies are moving toward the use of virtual hardware. These are software imitations of hardware that expedite the development and testing on a simulated target before the silicon is available and, in many cases, in parallel with the development of the hardware.

In support of virtual environments, Parasoft C/C++test can be deployed in containers. This allows testers to build images with all the desired tooling in place, integrate it into their CI/CD pipeline, then perform static analysis, unit testing, and code coverage. Teams can also find a Parasoft C/C++test image ready for deployment on DockerHub/Parasoft.

Service Virtualization

Service virtualization simulates all the dependencies needed by the device under test in order to perform full system testing. This includes all connections and protocols used by the device with realistic responses to communication. For example, service virtualization can simulate an enterprise server backend with which an embedded system communicates. Similarly, virtualization can control simulate a dependent system, like backend databases or data acquisition and control.

Service Level Testing of Embedded Software

Developers can build integrations earlier, stabilize dependencies, and gain full control of their test data with Parasoft Virtualize. Teams can move forward quickly without waiting for access to dependent services that are either incomplete or unavailable. Companies can enable partners to test against their applications with a dedicated sandbox environment.

Parasoft SOAtest delivers fully integrated API and web service testing tools that automate end-to-end functional API testing. Teams can streamline automated testing with advanced functional test creation capabilities for applications with multiple interfaces and protocols.

SOAtest and Virtualize are well suited for network-based, system-level testing of various types, including the following.

» Comprehensive protocol stack that supports HTTP, MQTT, RabbitMQ, JMS, XML, JSON, REST, SOAP, and more.

» Security and performance testing during integration and system testing with integration into the existing CI/CD process.

» Fuzzing and penetration testing.

» End-to-end testing that combines API, web, mobile, and database interactions into virtual test environments.

SUMMARY

The strategies, processes, or environments used to test software are referred to as software testing methodologies. Agile and waterfall are the two most popular SDLC methodologies, and testing is very different in these two environments. However, within these different approaches there are common types of testing such static analysis, unit, integration, system, and acceptance testing.

Static analysis solutions, like Parasoft C/++test, prevent and detect a wide range of error and warning classes, such as variable initialization, memory management, and memory overflow errors. They support critical development activities such as code inspections and coding standard enforcement.

Parasoft C/C++test automates many of the tedious and error prone activities needed for unit and integration testing. Reporting and analysis is provided by Parasoft DTP, which collects all of the results and details of the automated tests. Parasoft SOAtest provides system-level test automation, especially for network-connected applications and devices.

TAKE THE NEXT STEP

Talk to an expert to learn more about software testing methodologies and the most effective automated software testing approach for your development team.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives — security, safety-critical, Agile, DevOps, and continuous testing.