PARASOFT®

WHITEPAPER

How to Get Started Modernizing Government Legacy Systems

OVERVIEW

Legacy systems can be difficult to manage because there's often no active development team or product owner. This means there is no institutional memory for core technical elements. When these systems require a technology refresh or update to address major security flaws, they can be incredibly challenging to modernize.

Understanding and modernizing a legacy application is often expensive and high risk. Government furnished equipment (GFE) and government furnished information (GFI) come in an “as is” state. The poorer the state, the more risk contractors build into the bid, increasing the cost. In most cases, any modernization team will be new staff with no background in the previous development effort.

Documentation, if it exists, is in the last state that was afforded. Too often, as contractors address small fixes, documentation doesn't get updated because the fix is all that's afforded. This drift continues until a major issue arises that requires dramatic action to resolve. The problem deepens when the documentation doesn't match up with the current configuration of the system.

Once the team is selected, a typical application modernization effort begins with a transition plan. Application reconnaissance is a high skill and therefore expensive task. The incoming team reads any documentation and brings in senior developers to deconstruct the code base and the system. They look at the artifacts in the source code management system, assess the system code on the running server, and use log events to try to figure out what portions of code are being used and how they are related to key user experiences.

Along the way, the team notes where the map doesn’t match the landscape and where the documentation is not correct. Most importantly, they try to make sure that when they do make changes, the system still meets all original functionality and outputs remain consistent for customers.

This assessment can take weeks and still not be right. Key customers who don't engage with the system regularly or engage at longer intervals might not show up as noteworthy in logs. If no regression tests exist, there are significant risks:

- Any changes could break user experiences for core constituencies.
- Unknown customers may come out of the woodwork after the team turns off legacy functions.

ESTABLISH PROCESS CONTROL

The government management team needs to establish and strictly govern the following:

- Application source control
- Application build process control
- Enterprise test process control

These foundational process control elements become the core GFE/GFI. The controls, along with the key actionable elements described in the following sections, will be the complete GFE/GFI that the government archives in between periods of contractor maintenance.

You might have nonexistent or limited information to start with, but to achieve success it's paramount to first understand the legacy system and then control it.

UNDERSTANDING LEGACY SYSTEMS & APPLICATIONS

To understand legacy applications is to know:

- What they're comprised of—the underlying code and configurations.
- How they're built.
- How they operate from both a functional and nonfunctional perspective.

Once teams understand the details, they can control and archive the information. When new contractors step in with a focus on evolving the applications, they can more quickly reconstitute the application along with the ecosystem in which it operates.

Each of the following sections speak to this understanding while providing key insights into AI-enabled test automation solutions available to help implement the modernization of legacy applications as a manageable and sustainably cost-effective venture for the government.

CODEBASE SECURITY, SAFETY & QUALITY: STATIC ANALYSIS

Static analysis scans provide in-depth views into the workings of the code from a structural and maintainability perspective and from a security and safety perspective. A security and safety analysis looks for potential runtime security vulnerabilities and safety concerns. It does this by semantically executing a parse tree of the integrated application and playing "what if," looking for buffer read/write issues, memory infractions, database infractions, and much more. All of which could manifest in a production environment as potential catastrophic failures.

Pick a variety of well-established industry standards and guidelines to adhere to for security, safety, and quality, including MISRA, OWASP, CWE, and more. It's good practice to run multiples and even several static analysis tools as each vendor does things differently and may have strengths others do not.

Teams should incorporate static analysis as part of a fully automated DevSecOps CI/CD process, collecting ongoing compliance and trends reporting over time.

THE FUNCTIONAL GOOD, BAD, & UGLY OF YOUR CODEBASE: BUILD A POOR MAN'S REGRESSION SUITE

From the source code, create a suite of so-called "poor man's regression tests" for all code in the application. An automated process creates an extensive set of tests to isolate low-level code functionality and sets the stage for monitoring code changes going forward. This is done by taking the signatures of the various calls and automatically creating test cases to exercise with extreme conditions. The suite of tests allows you to test any changes going forward as well as stub out and isolate specific code segments. The process locks down the most granular, lowest level pieces of the application.

Next, run the suite of tests. Some may blow up, some may behave. Some tests are not relevant.

Based on expectations, the test team verifies the expected results, asserting these to be correct. These asserted tests become the validated regression tests.

If you give the code base to a new contractor or to another contractor, you can track new outputs against past outputs. As you evolve the code, ask:

- Did my change cause the test to fail?
- Does my test need to change based on how I want the functions to evolve?

Also, as you added this last code, the new team hopefully added a few unit tests. This is a very brute force approach, but it puts a stake in the ground. Before you can make decisions on a test strategy, you need to know where you are.

As mentioned, teams must assert and lock down all tests for correctness to monitor for potential future changes. Teams should incorporate tests as part of a fully automated DevSecOps CI/CD process.

THE CURRENT EXTENT OF TESTING VIA CODE COVERAGE: CODE NOT COVERED IS CODE NOT TESTED

Attach to the build process any existing unit testing, low-level integration testing, and application testing suites of tests. Solutions for code coverage can instrument the code and monitor the execution of any of the test runs to collect and combine the coverage results into a single truth of what code has and has not been tested.

The solution monitors and collects coverage metrics such as statement, line, decision, path, and MC/DC coverage. Understanding what code has been executed reveals areas that require more testing, the potential injection of unused code, the lack of tested requirements, and much more.

Note that it's not good practice to include poor man regression suites into this phase. You should only create poor man suites to establish a baseline of functional knowledge and monitor change going forward. Any code coverage collected from these runs would not have real-world value and as such would pollute valid coverage insights.

This entire process is building up the knowledge base about the codebase and puts the proverbial stake in the ground. A known starting point is paramount on the journey to evolve your system.

Collecting code coverage is part of a fully automated DevSecOps CI/CD process. Code coverage can and should be collected for all testing practices from unit testing through lower-level integration testing, component testing, application testing, and even system level testing. Coverage results should be collected, merged, and reported for cumulative coverage results.

MANAGING THE FUNCTIONAL BIG ROCKS: APIS, SERVICES, & INTEGRATION ENDPOINTS

Moving into the world of APIs, the message-based communication to/from applications, service and microservice layers as well as cross-domain endpoint integrations. These are higher level functional integration tests that assert capabilities tied back to key system requirements.

Leveraging the identification of API nodes along with the message traffic to/from them, an API testing solution can help automate the creation of a real-world, not poor man's, API functional regression testing framework.

Modern API testing solutions can set proxies between any key communication nodes in a system. These proxies collect traffic and leverage AI-enabled test suite generation capabilities to intelligently replace traditional playback testing with parameterized test suites. The parameterized test suites provide the flexibility to subsequently run tests driven by a database, Excel, and other data configuration mechanisms.

Parameterization provides the ability to execute both positive and negative testing runs to test functionality in expected and unexpected circumstances. This increases the quality of the system under test.

All tests can be asserted for correctness and locked down to monitor for potential future changes and incorporated as part of a fully automated DevSecOps CI/CD process. The underlying code comprising the API layers being tested should be instrumented to collect any required code coverage metrics as well. Test results and coverage results should be collected and reported on for regular review and analysis.

Leveraging the same message traffic recording capabilities as described, API testing solutions enhanced with AI and machine learning (ML) can create virtual services, like mocks. These robust yet lightweight simulations allow for easier integration of those layers that may be constrained because of external development constraints (another contractor's responsibility), yet-to-be developed systems, multi-version requirements, test environment multi-tenant conflicts, and more. These constraints can cause inability to access functionality, delays in test configuration setup, destruction of test data by co-tenants, cost overruns, and more.

Virtual services are intelligent with the ability to:

- Understand the use case for which they were created.
- Respond as if they were a real service thanks to timing mechanisms.
- Provide intelligent responses based on dynamic data inputs.

AI/ML capabilities enhance the creation of these assets by analyzing traffic and data patterns over time.

MANAGING THE FUNCTIONAL PEBBLES: SERVICE & MICROSERVICE OPERATIONS REGRESSION TESTS

The same traffic recording for API test generation and virtual service generation can be applied at the finest granular API operational levels to provide an extensive regression testing suite for monitoring change at the microservice and service operations level, isolating expected capabilities of a specific service operation.

Creation of such granular regression suites can be enhanced by leveraging system traffic monitoring solutions, such as those that can log system-wide message traffic. The message logs can help accelerate the creation and controlling of such regression testing suites.

Again, teams should incorporate regressions suites as part of a fully automated DevSecOps CI/CD process along with reporting for functional testing trends analysis.

PERFORMANCE ABILITIES THROUGH REUSE OF EXISTING TEST SUITES

Once you establish key API test suites, then you can leverage these same suites to establish load and performance metrics for your system. Without any scripting necessary, you can spin up virtual users configured to bang on your system from any API vantage point and configuration with dynamic test data inputs, over time, injected specifically when required. Response times, memory, and network utilization, as well as many other system level metrics, may be graphed, reported, and asserted to meet specific SLAs as established by your requirements.

You can incorporate load and performance suites as part of your fully automated DevSecOps CI/CD process for reporting and ongoing trends analysis.

USER EXPERIENCE: WEB UI TESTING

Not all systems will have user interfaces (UI) associated with them, but many will. With Selenium being a major thrust at automating user functional testing for web-based applications, it's a good idea to build out a regression suite of user workflows.

Many organizations will already have Selenium test suites in place, but many others will not. For those with an entire focus on manual user testing, the ability to leverage automated Selenium test generation—using the same traffic recording techniques described earlier—is a great productivity accelerator.

Selenium tests, while powerful, are notorious for being extremely brittle to change. Leveraging AI/ML to self-heal broken Selenium tests at runtime reduces the triage timelines and heartache necessary to maintain any tests that may be involved during the journey in locking down the overall state of the system.

ADDITIONAL INSIGHTS

When it comes to legacy modernization you won't be successful on your journey without knowing where you're starting from. You must first get your arms around the current state of how things operate. You must first understand. This can be a journey in and of itself if there are limited testing processes in place to begin with. Test automation is key to establishing the frameworks for accelerating the understanding.

ENHANCE TESTING WITH AI & ML

To help in this journey, AI and ML are both key foundations to accelerating test generation, execution, maintenance, and more. AI can be folded into all layers of testing.

At the unit test level, AI automatically generates the missing unit test cases. At the API test layer, AI tracks patterns and relationships in API calls and automatically helps developers create advanced scenarios to test the system as well as a data model of the system parameters.

The AI also helps users by learning from Selenium tests and then making those tests self-healing. As developers change code, AI technologies modify and update the Selenium tests. AI also creates flags for change in performance associated with application updates.

MOVE TO CI/CD

This modern test infrastructure should be a part of a modern delivery process. A CI/CD process that controls and manages all elements of application assembly, testing, and release. The entire CI/CD infrastructure, combined with the underlying code and automated test suites, defines your legacy system going forward. It is this entirety that is archived for future use. It is this entirety that is reactivated for future evolution. Its execution and reporting are the definition of that legacy system.

These modern testing practices and their results are what enable this *understanding*. Eventually this understanding will move into the evolution of the application, its modernization. It's the traceability from all testing practices, directly to the underlying source code, that enables the acceleration of modernization efforts.

OPTIMIZE TESTING WITH TEST IMPACT ANALYSIS

Modernization is evolution. This means code will be added or updated. When dealing with complex applications and systems of systems the total number of tests across varying practices can be overwhelming. Test impact analysis (TIA) determines the minimum test set execution required to exercise just the updated code.

TIA saves significant time and operational dollars tied to the constant execution of DevOps pipelines. In addition, TIA's fine-grained understanding of the application provides greater insight and control of the ongoing evolution of the modernized application.

As a new team adopting this well-understood system, the modernizing efforts become easier, faster, and more cost-effective because you're using the same solutions as those used for understanding.

TAKE THE NEXT STEP

Email governmentsolutions@parasoft.com for more information about modernizing government legacy systems.

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.