Quality Assurance Surveillance Plan

A tool to help you know you're getting high-quality software from your vendor.

High-quality software is:

1. Tested
2. Properly styled code
3. Accessible
4. Deployed
5. Well-documented
6. Secure
7. Based on user research

The QASP is a tool that allows for good oversight of a software development project without affecting the development team's velocity. The aim is to create a smooth working process that's more informative than a simple written report or traditional IV&V reviews. It's also valuable to provide vendors your expectations and how you'll be measuring their performance, so they can maintain high-quality work throughout the project.

The elements of the QASP increase confidence that you're getting the software you need, and that it's high-quality software that you'll be able to extend and maintain over time. It's also a communication tool that helps you identify exactly what things you need to discuss with your vendor. One of the QASP metrics is coming up a bit low? That's something you can talk about.

What does that mean?

TESTED CODE

"Tested code" refers to automated tests, not the kinds of manual tests you might be used to with user-acceptance testing. Automated tests are pieces of code that run your application and make sure it works as expected. These tests can range from small tests that check one small piece of internal functionality ("unit tests") all the way to tests that verify a page changes correctly when a person interacts with it ("integration tests"). A good set of tests can completely replace user-acceptance testing.

Why it matters
Tests give you confidence that the code works as expected, not just today, but for the future as well. If the code's behavior changes in unexpected ways, good tests can catch that change early and prevent a bug from ever reaching end users. Even though writing tests can slow down development, it's a good investment because good tests reduce the risks associated with making changes in the future. Building automated tests right into your development process also removes user-acceptance testing as a hurdle to releasing your app because at every step, every day, you have confidence that your tests are passing — and if they're not, you know exactly what to discuss with your vendor.

How to do it

Your vendor will write the automated tests alongside the app code. The government product team's job is to look at the results of the tests and make sure they're satisfactory. Many test-runners will show you an easy-to-read summary with green checkmarks for tests that pass and red exes for any that fail. If you also adopt a continuous integration platform, you can have it run the tests for you automatically on every change to the code, and you can just periodically look at a website to see the current state of your tests.

PROPERLY STYLED CODE

Code style refers to how the code itself is written. Does it conform to best-practices? Does it use common patterns? Is it consistent throughout? Individual developers have their own, such as preferring to use 2 spaces for indentation versus 4 spaces or a tab character. This requirement is about standardizing on all those kinds of things so the code is the same everywhere.

Why it matters

Properly styled code is easier for a team to work on because it consistent. Developers can more easily move around the code to work on different parts because it's all very similar. New developers on the project can also more easily pick it up because it's consistent, and because it's using common patterns and best practices, it will probably look similar to other code they've seen before.

How to do it

Code style is checked with an automated code scanning tool called a linter, usually combined with a manual code review to find any higher-level logic issues that linters don't catch. There are other tools – static analysis scanners – that can identify other code style and quality issues, such as sections that are overly complex or too long. These issues indicate code that is likely to be hard to maintain over time, so when they show up, it's good to bring it up with the vendor and make a plan to fix it.

ACCESSIBLE

Accessibility is about making sure people with varying levels of ability are able to use it. For example, can a non-sighted person use your website? Can a colorblind person understand the information in your application? The Americans with Disabilities Act requires that federal, state, and local governments be accessible to people with disabilities, and the Department of Justice has repeatedly said that includes websites. So it's not just the right thing to do, it's also a legal requirement. Section 508 of the ADA was recently updated to point to the Web Content Accessibility Guidelines (WCAG) version 2.1 as the federal standard for accessibility compliance, so that's what we use.

Why it matters

Accessible applications strive to ensure that all of your users can use your apps. For example, an accessible eligibility and enrollment website would allow a non-sighted user to sign up for benefits directly rather than having to call and speak to someone in the office. It empowers all of your users, not just your sighted, hearing, and able-bodied users. "Situational disabilities" — such as a carpal tunnel flare-up that inhibits the use of a mouse, or misplacing your glasses requiring you to increase the font size of a website — will affect almost all of us at some point in time. Building accessible applications means that even during temporary setbacks, users will be able to accomplish their goals with your app.

It's also the law.

How to do it
Testing for accessibility requires specialized knowledge, but it's worth the investment. There are automated tools that can check some aspects of your app, but some of it has to be tested manually by putting your hands on it and verifying that it behaves as expected. For example, there's no automated tool that can listen to a screen reader and confirm that it says the right things, so someone will have to do it manually. There are vendors that can do the accessibility testing for you, or your own staff can get trained on it. The Department of Homeland Security offers a free training program called "Trusted Tester," for example.

DEPLOYED

Code is just a bunch of text. Until it's deployed, there's no way to interact with it and make sure it's going to meet your needs. And in 2019, there's also no reason deployment needs to be a complicated, days-long process. Typical one-step deployments today average around 7 minutes and don't require any human intervention once kicked off. Most modern product teams will have several deployment environments for various needs.

Why it matters
A deployed app is an app you can test, demo, and give to users. If you can't deploy, you don't really have an app - you just have a pile of code. Multiple environments are important too. For example, a staging environment gives you a chance to see how your app is evolving before it goes out to users, and being identical to production gives you confidence that it'll deploy successfully into the production environment in the future. With modern computing platforms like Infrastructure as a Service (IaaS) or Platform as a Service (PaaS), it's possible to quickly and easily create an arbitrary number of environments to meet your needs.

Automated deployments are accomplished with the use of scripts. These scripts make the deployment perfectly repeatable. When a person has to manually perform each step of a deployment, they will invariably forget steps or do them incorrectly from time to time. It's human nature - even with checklists, we make mistakes. A scripted, automated deployment, however, will work the same, every time, and will stop itself if it encounters errors. That will increase your confidence that future deployments will go smoothly and that you can release bug fixes and new features quickly and easily.

How to do it
Your vendor should take care of most of the work of setting up the deployment. You and your IT staff will need to help the vendor get access to your infrastructure so they can deploy. You can verify that the deployments are working by using the app - it should be the most recently-deployed version. You can also lean on your internal IT staff to take a look at the deployment script or process to verify that it's simple to start.

WELL-DOCUMENTED

Good documentation helps developers understand the code, whether they're the same developers that originally wrote it or a new team picking it up later. Explaining assumptions, logic flows, and expectations can save developers a lot of time later. Documentation can refer to "comments" directly in the code indicating what it does and system diagrams showing how various pieces connect to each other and to other systems.

Why it matters
Solid documentation will help future developers pick up the code and continue working on it, which makes it easier for the government to change vendors if necessary, or shift to in-house maintenance and new development.

How to do it
Ensure that documentation is updated as major features are added or updated, and look out for too much technical jargon. You might not understand what all of it means, but you should be able to read it and make some sense of what it means at a higher level. If you get completely lost, that's a great time to talk with your vendor to have them explain, and maybe even revise the documentation.

SECURITY

Security is a broad topic, and it's largely about protecting the data that your system contains. The fundamental goal of security is to make it difficult enough for someone to access your data that it's not worth the effort. That means for systems with low-value data, the security does not need to be as strong: an attacker might be willing to spend 6 months getting access to credit card numbers, but probably wouldn't spend more than a few minutes trying to access your grocery list.

Why it matters
Good security matters because it means your data is more likely to remain safe, and by extension, your users are likely to be safer too. A system being breached by an attacker doesn't just result in bad headlines. It can also make critical services unavailable to the people who need them, and it can result in people's personal information being used in ways that harm them, such as identity theft.

How we do it
Software security cannot be perfect, but there are tools to help us identify vulnerabilities before anyone can exploit them. There are "static" and "dynamic" scanning tools that can run well-known sets of exploits against your app to see what happens and let you know if you're vulnerable to them and how. There are also dependency scanners that can check all of the external libraries your app uses to see if they have vulnerabilities. All of this can be automated and the results can show up right alongside your code tests, making it easy to quickly see your security health and identify what you need to talk with your vendor about.

BASED ON USER RESEARCH

Usability research is the process of interviewing and observing users to understand what problem they're trying to solve with your software, their conceptual model of how to solve it, and how they use your app for their needs. It is the cornerstone of designing good software, guiding what to build, how, and when.

Why it matters
If you don't know what your users need, how they think about the problem, or how they interact with your app, you can't know what to build for them. In fact, by guessing at what you think they need rather than learning what they need through observation, you could very well end up building an experience

that is unhelpful, difficult to use, or confusing. To create software that meets user's needs, you must base your app on research.

How to do it

Usability research is most effective when it starts at the very beginning of the app's or feature's lifecycle. Ensure you're testing during new feature development with actual end users of the system; don't wait until you think you have something completely finished. Often, what you learn by testing early will influence how you choose to prioritize and build parts of a feature.

When your vendor conducts usability research, ask to ride along. You don't need to say anything or be involved, just be a silent observer. You can also ask for their research synthesis when they're finished, and incorporate what you learn into your planning for upcoming work.

List of Deliverables with Quality Assurance Surveillance Plan (QASP)

The following chart sets forth the performance standards and quality levels the code and documentation provided by the Contractor must meet, and the methods the [agency] will use to assess the standard and quality levels of that code and documentation.

Deliverable	Performance Standard(s)	Acceptable Quality Level	Method of Assessment
Tested Code	Code delivered under the order must have substantial test code coverage. Version-controlled [Agency] GitHub repository of code that comprises product that will remain in the government domain.	Minimum of 90% test coverage of all code. All areas of code are meaningfully tested.	Combination of manual review and automated testing
Properly Styled Code	GSA 18F Front-End Guide – https://frontend.18f.gov/#js-style	0 linting errors and 0 warnings	Combination of manual review and automated testing
Accessible	Web Content Accessibility Guidelines 2.1 AA standards	0 errors reported using an automated scanner and 0 errors reported in manual testing	https://github.com/pa11y/pa11y

Deployed	Code must successfully build and deploy into staging environment.	Successful build with a single command	Combination of manual review and automated testing
Documented	All dependencies are listed and the licenses are documented. Major functionality in the software/source code is documented. Individual methods are documented inline in a format that permit the use of tools such as JSDoc. System diagram is provided.	Combination of manual review and automated testing, if available	Manual review
Secure	OWASP Application Security Verification Standard 3.0 (or similar tool)	Code submitted must be free of medium- and high-level static and dynamic security vulnerabilities	Clean tests from a static testing SaaS (such as Snyk or npm audit) and from OWASP ZAP (or similar tool), along with documentation explaining any false positives

User research	Usability testing and other user research methods must be conducted at regular intervals throughout the development process (not just at the beginning or end).	Research plans and artifacts from usability testing and/or other research methods with end users are available at the end of every applicable sprint, in accordance with the contractor's research plan.	[Agency] will manually evaluate the artifacts based on a research plan provided by the contractor at the end of the second sprint and every applicable sprint thereafter.