MESC 2019 Workshop:

Practical tips to effectively lead digital transformations and modernize legacy systems

August 19, 2019

Robin Carnahan, robin.carnahan@gsa.gov
Randy Hart, randy.hart@gsa.gov
Greg Walker, michael.walker@gsa.gov
Amy Ashida, amy.ashida@gsa.gov

18F, Technology Transformation Services, General Services Administration

Introduction: This template is meant to assist agencies with writing a solicitation agile development services. Please read the prerequisites for working this way and see an example of this method in practice.

How to use: Tailor to your agency's needs and delete all blue helper text. If you're interested in working with 18F, get in touch at inquiries18f@gsa.gov.

[Agency/State]

[Program]
Statement of Objectives for [Product]

1.0 Background and Purpose

- *If you have additional supporting information that might be helpful to contractors, we suggest attaching it as an Appendix. We try to keep these solicitations brief, typically 20 pages or less.*

1.1 Background

- *Include a brief summary of your agency's mission.*
- *Include a brief summary about how your team fits within the agency.*
- *Write in plain language; assume that potential contractors have never heard of your agency.*

1.2 Problem

- *The software product will be developed to solve a problem on behalf of users.*
- *Include a concise summary of the problem that needs to be solved.*
- *Call out pain points with existing implementation, if appropriate.*

2.0 Scope

- *The scope should be defined by the product vision to enable flexibility in developing the solution.*
- *Include language to ensure that it's clear that this project is open by default.*
- *Make sure the solicitation states what the contractor is responsible for vs. what the government is responsible for.*

2.1 Product Vision

- *Product vision is the reason you're creating the product; the overarching goal the team is aiming for.*
- *Product visions should be clear, concise, informative, and inspirational.*
- *When drafting, consider:*
    - *Who is affected?*
    - *What is the problem?*
    - *How are we helping?*
    - *What is the outcome?*
- *Do not include any implementation details.*

The [agency] intends that the software delivered under this task order will be committed to the public domain. The Contractor will have to obtain [agency] permission before delivering software under this task order that incorporates any software that is not free and open source. The Contractor must post all developed code to a [Git] repository designated by the [agency].

2.2 Anticipated Period of Performance, Budget, and Ceiling Price

- *We recommend including the anticipated budget in the solicitation. Contractors have found it easier to right-size their proposed team when they have a sense of what the budget will be.*

- *We recommend the length of the contract not exceed three years. Anything longer could lead to too much dependency on a single contractor.*
- *When building your budget estimate, use Contract Awarded Labor Category (CALC) to see labor rates typically charged to the Government by labor category.*
- *Depending on your scope of work, an agile development team of 4-8 people will cost around $1.5 - 2M per year.*
- *If your contracting officer is not comfortable including the budget in the solicitation, an alternative is to put forward the anticipated size of the team. You will get better bids when you are up-front with contractors. Otherwise, there is too much guess work and potential for pricing gamesmanship.*

The not to exceed ceiling on this contract will be $W million for the X period of performance and up to an additional $Y million for Z option periods.

3.0 Objectives

3.1 Backlog

- *Include the draft product backlog of user stories.*
- *It's important to note that these are primarily to give context to the contractors' for the product that they will build — but these stories will change as the team begins working with your agency and the users.*
- *More information on drafting user stories can be found here.*

The set of preliminary user stories set forth below will be the starting point for the development of software to be provided under this [contract action]. These preliminary user stories are provided only for illustrative purposes, and do not comprise the full scope or detail of the project. The [agency] expects that the Contractor will work closely with the Product Owner to perform regular user research and usability testing and to develop and prioritize a full gamut of user stories as the project progresses.

Individual user stories may be modified, added, retracted, or reprioritized by the [agency] at any time, and the [agency] expects that the user stories will be continuously refined during the development process.

- As a <type of user> I want <some goal> so that <some reason>.

3.2 List of Deliverables with Quality Assurance Surveillance Plan (QASP)

- *We recommend using the following QASP to ensure all code that is submitted is tested, properly styled, accessible, deployed, documented, and secure. This is the checklist that the technical team will be responsible for delivering against every sprint.*
- *The QASP may be included in the solicitation or in a stand-alone attachment.*

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	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	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, 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.

4.0 Contract Place of Performance and Contract Type

- *List place of performance. We recommend allowing performance to be off-site/distributed if at all possible because you'll get access to the best talent and best prices.*
- *For contract type, we recommend a time and material contract with a not-to-exceed ceiling. Time and material contracts increase team flexibility and avoid the gamesmanship inherent in firm fixed pricing. The not-to-exceed ceiling limits agency risk.*

The Contractor may choose the location(s) from which to perform the required software development services. [provide agency's core working hours if you want to make sure the development team accommodates that schedule].

5.0 Operating Constraints (Non-functional Requirements)

5.1 Environment

- *List non-functional requirements that the contractor should be aware of such as development environment, integration with other systems, use of the United States Web Design System, etc.*

5.2 Personnel Skills and Knowledge

- *We recommend only requiring two key personnel (most often, project manager and technical lead) to avoid bait-and-switch and also to prevent the bidding companies from having too many people waiting on the bench.*

Key Personnel – The Contractor must designate both a Project Manager (PM) and a Technical Lead as Key Personnel for this project. The PM will be a direct liaison to the [agency] product team, and will be responsible for the supervision and management of all of the Contractor's personnel. The Technical Lead must have a full understanding of the technical approach to be used by the Contractor's development team and will be responsible for ensuring that the Contractor's development team follows that approach.

5.3 Special Clauses

- *Include any special clauses, including data rights and invoicing instructions.*
- *See here for more information on how and why 18F uses open source software.*

Data Rights and Ownership of Deliverables – the [agency] intends that all software and documentation delivered by the Contractor will be owned by the [agency] and committed to the public domain. This software and documentation includes, but is not limited to, data, documents, graphics, code, plans, reports, schedules, schemas, metadata, architecture designs, and the like; all new open source software created by the Contractor and forks or branches of current open source software where the Contractor has made a modification; and all new tooling, scripting configuration management, infrastructure as code, or any other final changes or edits to successfully deploy or operate the software.

To the extent that the Contractor seeks to incorporate in the software delivered under this task order any software that was not first produced in the performance of this task order, the [agency] encourages the Contractor to incorporate either software that is in the public domain, or free and open source software that qualifies under the Open

Source Definition promulgated by the Open Source Initiative. In any event, the Contractor must promptly disclose to the [agency] in writing, and list in the documentation, any software incorporated in the delivered software that is subject to a license.

If software delivered by the Contractor incorporates software that is subject to an open source license that provides implementation guidance, then the Contractor must ensure compliance with that guidance. If software delivered by the Contractor incorporates software that is subject to an open source license that does not provide implementation guidance, then the Contractor must attach or include the terms of the license within the work itself, such as in code comments at the beginning of a file, or in a license file within a software repository.

In addition, the Contractor must obtain written permission from the [agency] before incorporating into the delivered software any software that is subject to a license that does not qualify under the Open Source Definition promulgated by the Open Source Initiative. If the [agency] grants such written permission, then the Contractor's rights to use that software must be promptly assigned to the [agency].

6.0 Instructions and Evaluation

6.1 Submission Instructions

- *Most agencies have specialized instructions for receiving bids. Include those here, such as submission format, due date, where to send inquiries, etc.*

6.2 Instructions for Proposals

- *We suggest using similar language to the text below for technical submissions and evaluation criteria.*
- *We recommend asking for samples of previous work (Git repositories) to demonstrate performance.*
- *We recommend conducting interviews to get a good sense of which proposed team will work best with the government team.*

Technical Submissions

Technical submissions must consist of a technical proposal of no more than four (4) pages, a staffing plan of no more than three (3) pages plus resumes and signed letters of intent to participate, and references to one or more source code samples. Technical submissions may also include user research plans and design artifacts of no more than 30 pages combined. Technical proposals and staffing plans must be submitted using 12-point type.

The technical proposal must set forth the Offeror's proposed approach to providing the services required, including the base software (if any) and programming language(s) the Offeror proposes to use. The technical proposal must also make clear that the Offeror understands the details of the project requirements. The technical proposal must also identify potential obstacles to efficient development and include plans to overcome those potential obstacles. The technical proposal must also include a description of the Offeror's plans, if any, to provide services through a joint venture, teaming partner, or subcontractors.

The staffing plan must set forth the Offeror's proposed approach to staffing the requirements of this project, including the titles of each of the labor categories proposed and proposed level of effort for each member of the Offeror's development team. The staffing plan must also identify the proposed Project Manager and proposed Technical Lead by name, and include a resume for each. Those resumes must include a brief description of the experience and capability for each individual, but cannot exceed one (1) page in length each. Offerors proposing Key Personnel who are not currently employed by the Offeror or a teaming partner must include a signed letter of intent from the individual proposed as Key Personnel that he/she intends to participate in this project for at least one (1) year. The staffing plan must also set forth the extent to which the proposed team for this project was involved in the development of the source code referred to in the next paragraph.

The staffing plan must set forth and explain the extent to which the Offeror will provide individuals with experience in at least each of the following areas [tailor for your project]:

- Agile development practices
- Automated (unit/integration/end-to-end) testing
- Continuous Integration and Continuous Deployment
- Refactoring to minimize technical debt
- Application Protocol Interface (API) development and documentation
- Open-source software development
- Cloud deployment
- Open-source login/authentication services

- Product management and strategy
- Usability research, such as (but not limited to) contextual inquiry, stakeholder interviews, and usability testing
- User experience design
- Sketching, wireframing, and/or prototyping, and user-task flow development
- Visual design
- Content design and copywriting
- Building and testing public facing sites and tools

The references to one or more source code samples must be either links to Git repositories (either credentialed or public) or to equivalent version-controlled repositories that provide the [agency] with the full revision history for all files. If an Offeror submits a link to a private Git repository hosted with GitHub, the [agency] will provide the Offeror with one or more GitHub user identities by email, and the Offeror will be expected to promptly provide the identified user(s) with access to the private Git repository.

The source code samples should be for projects that are similar in size, scope, and complexity to the project contemplated here. The source code must have been developed by either (i) the Offeror itself, (ii) a teaming partner that is proposed in response to this RFQ, or (iii) an individual that is being proposed as Key Personnel for this project. The [agency] would prefer that the source code samples have been for recent projects involving teams of approximately X - Y [tailor for your project] Full-Time Equivalent (FTE) personnel.

If the references to source code samples provided do not include associated references to user research plans and design artifacts demonstrating how ongoing user research was incorporated into the project, then the Offeror must submit a user research plan and design artifacts relating to at least one (1) of the source code samples provided.

Price Submissions

Price submissions must set forth a single dollar amount that represents the Offeror's estimate of the total cost to the [agency] for the development services and travel expenses required for [period of performance]. [Instructions on providing price proposal, typically an excel workbook]. The [agency] expects that the labor categories and staffing levels set forth by the Offeror in the Excel workbook will be consistent with the Offeror's staffing plan.

The Contractor will be compensated at loaded hourly rates. The [agency] intends to evaluate proposals and award based on initial proposals, and therefore the Offeror's initial proposal should contain the Offeror's best terms.

Interviews

The Offerors with the most highly rated written submissions will each be invited to participate in an interview as part of the evaluation process. Each interview will be conducted remotely via video connection and/or teleconference. The [agency] will communicate with certain Offerors to schedule the dates and times of interviews.

Each interview will include an unstructured question and answer session, during which Offerors will be asked questions about the technical aspects of their proposal and their approach to software development. The [agency] expects these interviews will assist the [agency] to assess the technical abilities of the proposed development team and to better understand the proposed technical approach described in the Offeror's written submission. Both of the Offeror's proposed Key Personnel must participate in the interview.

Interviews will not constitute discussions. Statements made during an interview will not become part of the agreement.

Basis of Award and Evaluation Factors

Each submission received by the [agency] will be evaluated for technical acceptability. Submissions that are determined to not be technically acceptable after the Offeror has been given the opportunity for a clarification will not be evaluated further.

Quotes must be realistic with respect to technical approach, staffing approach, and total price. Quotes that indicate a lack of understanding of the project requirements may not be considered for award. Quotes may indicate a lack of understanding of the project requirements if the staffing plan does not use a realistic mix of labor categories and hours, or if any proposed hourly labor rates are unrealistically high or low.

The [agency] will evaluate quotes that are technically acceptable on a competitive best value basis using a trade-off between technical and price factors. Technically acceptable submissions will be evaluated based on four (4) evaluation factors. These factors are

(1) technical approach, (2) staffing approach, (3) similar experience, and (4) price. The three (3) technical, non-price evaluation factors, when combined, are significantly more important than price. The [agency] may make an award to an Offeror that demonstrates an advantage with respect to technical, non-price factors, even if such an award would result in a higher total price to the [agency]. The importance of price in the evaluation will increase with the degree of equality between Offerors with respect to the non-price

factors, or when the Offeror's price is so significantly high as to diminish the value to the [agency] of the Offeror's advantage in the non-price factors.

Technical Approach

In evaluating an Offeror's technical approach, the [agency] will consider (a) the quality of the Offeror's plans to provide the open source, agile development services required, including user research and design, (b) the extent of the Offeror's understanding of the details of the project requirements, and (c) the extent to which the Offeror has identified potential obstacles to efficient development, and has proposed realistic approaches to overcome those potential obstacles.

Staffing Approach

In evaluating an Offeror's staffing approach, the [agency] will consider (a) the skills and experience of the Key Personnel and other individuals that the Offeror plans to use to provide the required services, (b) the mix of labor categories that will comprise the Offeror's proposed development team, and (c) the Offeror's proposed number of hours of services to be provided by each member of the Offeror's proposed development team.

Similar Experience

In evaluating an Offeror's similar experience, the [agency] will consider the extent to which the Offeror has recently provided software development services for projects that are similar in size, scope, and complexity to the project described in this RFQ, and the quality of those services. In evaluating the quality of those services, the [agency] will consider, among other things, the revision history for all files in the source code samples provided. The [agency] will also consider the user research and design-related artifacts that were associated with the source code samples provided or submitted separately. In considering an Offeror's similar experience, the [agency] may also consider information from any other source, including Offeror's prior customers and public websites.

Price

In evaluating an Offeror's price, the [agency] will consider the total of the Offeror's estimated costs for the development services, and travel expenses proposed, for [period of performance]. This total amount will be [excel workbook].

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.

SECURE

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.