De-risking Custom Technology Projects

A handbook for retooling budgeting, procurement, and oversight of major government software projects, to reduce risk of failure and to return control to government

By Robin Carnahan, Randy Hart, and Waldo Jaquith

18F, U.S. General Services Administration

Updated and designed for print by

US DIGITAL RESPONSE

Table of contents

Introduction

Only 13% of large government software projects are successful.[1] In large part, that's because government software procurement processes are broken.

Government is increasingly reliant on modern software and hardware to deliver essential services to the public, and the success of any major policy initiative depends on the success of the underlying software infrastructure. Federal, state, and local government agencies all confront similar challenges, facing budget and staffing constraints while struggling to modernize legacy technology systems that are out-of-date, inflexible, expensive, and ineffective.[2] Government officials and agencies lacking basic knowledge about modern software development often rely on the same legacy processes that led to problems in the first place.

The American public deserves a government that provides the same world-class technology they get from the commercial marketplace. Trust in government depends on it.

This handbook is designed for executives, budget specialists, legislators, and other "non-technical" decision-makers who fund or oversee state government technology projects. It can help you set these projects up for success by asking the right questions, identifying the right outcomes, and equally important, empowering you with a basic knowledge of the fundamental principles of modern software design.

This handbook also gives you the tools you need to start tackling related problems like:

- The need to use, maintain, and modernize legacy systems simultaneously

[1] Projects valued at $6M or greater, in Europe and the United States, that were completed satisfactorily, on time, and within budget. From the Standish Group's "Haze," based on their CHAOS database. <https://www.standishgroup.com/sample_research_files/Haze4.pdf>

[2] Of the $90 billion in federal IT spending in FY2019, 80% was allocated for maintenance of legacy software, according to the GAO's June 2019 report, "Agencies Need to Develop Modernization Plans for Critical Legacy Systems." <https://www.gao.gov/products/gao-19-471> They write that inadequately-maintained legacy software leads to security risks, unmet mission needs, staffing issues, and increased costs.

- Lock-in from legacy commercial arrangements
- Siloed organizations and risk-averse cultures
- Long budget cycles that don't always match modern software design practices
- Security threats
- Hiring, staffing, and other resource constraints

This is written specifically for procurement of custom software, but it's important to recognize that commercial off-the-shelf software (COTS) is often custom and Software as a Service (SaaS) often requires custom code. Once any customization is made, the bulk of this advice in this handbook applies to these commercial offerings. (See "Beware the customized commercial software trap" for details.)

As government leaders, we must be good stewards of public money by demanding easy-to-use, cost-effective, sustainable digital tools for use by the public and civil servants. This handbook will help you do just that.

About the authors

We work for 18F, part of the Technology Transformation Services team at the General Services Administration (GSA). Collectively, the three of us have many years of experience in government procurement, software development, and state-level elective office.

In work funded by GSA's 10x, we spent a year meeting with state legislators, legislative fiscal staff, state budget officers, contracting officers, and gubernatorial policy advisors. This handbook came of learning from and teaching hundreds of people from dozens of states. We're grateful to the many people who contributed their time and knowledge throughout that process.

—Robin Carnahan, Randy Hart, and Waldo Jaquith

Basic principles of modern software design

A technology project's odds of success improve when the "non-technical" government leaders who fund and oversee it understand six basic concepts of modern software development: **user-centered design**, **Agile software development**, **DevOps**, **building with loosely coupled parts**, **modular contracting**, and **product ownership**. You don't have to be a technologist to understand these general concepts. Once you grasp them, it'll feel like you've gained a new super power, allowing you to spot the BS and cut through the jargon and technical detail, and stay focused on the basics of successfully guiding any software project.

User-centered design

All software development should be centered on the needs of the software's actual end users, the specific people who are expected to use it. These "end users" may be applicants for benefits, call center workers, case workers, other state employees or any of innumerable other groups.

Designing with and for users reduces project risks by ensuring the software is solving actual problems (as opposed to what a few stakeholders think the problems actually are). These problems are identified via a variety of research tactics, including interviews and testing for usability.

In user-centered design, all work is in the service of those end users' needs. That work is identified and prioritized in close and regular collaboration with end users, and is informed by, but not subservient to, any technical constraints. (That is, the goal of the work is to deliver value to users, which involves dealing with the realities of approved programming languages or server software, but work should never be omitted because of the perception that technical constraints would make it impossible.) The technical team and end users regularly review the work, as it is being performed, and the development work on the new software is not considered finished until those end users agree that their needs have been met. Designing with and for users reduces project risks by ensuring the software is solving users' problems.

In short, user-centered design says to do what actual, relevant humans need, and not what their boss's boss *thinks* that they need.

Agile software development

Detailed, long-term plans for major, custom software projects have long been the norm in government. But, as software engineers and policy makers have learned over the years, those plans are never correct. They need a lot of costly modifications, leading to requests for more money to pay for "change orders." It's time for government executives and budget officials to stop asking for detailed long-term plans, and instead to budget for software projects in a new way.

Planning an entire project upfront is known as "waterfall" development. Imagine planning a month-long family vacation of driving around the United States. Under waterfall, this would entail planning up front each day's agenda, including the route driven, booking every hotel room, pre-paying for every meal, pre-buying tickets for admission to attractions, etc. This would never work because things change, unexpected options come up, and no rational person would want to lock in every decision at the start of the journey when they don't know what the journey holds. Instead, most people would map out the general route to be taken and plan a few major stops—the specifics would be sorted out as they progressed along the way.

"Agile software development" refers to using this trip-planning methodology for building and modernizing software systems. Instead of relying on years of costly planning and "requirements gathering" before beginning to write actual software, Agile development projects are planned only in broad strokes, with a well defined description of the overall project goal and a strong preference for *just getting started*. A small, empowered, self-motivated team (usually 5-9 people, including developers, product managers, user researchers, writers, and/or security experts) is dedicated to accomplishing that goal, using user-centered design, working in two-week cycles to deliver some actual working software.

On day one, the team plans only what they'll do for the next two weeks. (The length of a project's cycles can be as brief as one week or as long as four weeks—two weeks is the most common.) Each task they'll work on is in the form of a "user story"—a specific user need revealed by user research.[3] The entire collection of user stories to be worked on is called the "backlog."

The team works on a selected group of user stories for two weeks and, at the end, the team reviews the work that they did, tests it with end users, and then plans the next two weeks by pulling more user stories from the backlog. Repeat. Each of these two-week cycles is referred to as a "sprint."

In the beginning, the software they produce may not seem like much (and may even be replaced by something else later), but it will gradually and systematically inform the project's technical approach and help the team sensibly integrate the project into an agency's existing legacy system.

Functioning software is delivered at the end of each sprint, without exception—fully-tested, fully-documented, ready to be used. In this way, value is delivered constantly, until the software is good enough to be rolled out for broad use. The team continues to work until they accomplish all of the goals or they run out of money, whichever happens first.[4]

---

[3] A user story reads in the form of "as a [role], I need [this thing], so I can [accomplish this]." For example, "as a social worker, I need case notes to be cached on my phone, so that I can access case notes in areas without mobile phone service." Every technical task is done in the service of addressing a user story.

[4] Stack Overflow's 2018 survey of 57,075 developers found that 86% of professional software developers use Agile; they stopped asking in subsequent years, presumably because of the uniformity of responses. And a 2015 study by Hewlett Packard found that "the vast majority of organizations [they] surveyed reported that today they primarily use Agile methods." The process described here is not extraordinary in any way.

The vendor is paid for their employees' time, not for a software system. Everything created by the vendor—software, documentation, research, designs, *everything*—is owned by government, delivered to government at the end of each sprint. Technology changes, government policies change, regulations change, laws change, and leadership's priorities change—any project that is planned in great detail up front will be unable to adapt to those changes, and will be at significant risk of failure, significant cost and deadline overruns, or costly "change orders."

By coupling Agile with user-centered design, a development team can constantly iterate toward solving the needs of end users in ways that would have been impossible to learn about up front.

In 2019, the U.S. Department of Defense's Defense Innovation Board released its Software Acquisition and Practices (SWAP) Study including a concept paper on "Detecting Agile BS" <https://media.defense.gov/2019/May/02/2002127286/-1/-1/0/DIBGUIDEDETECTING AGILEBS.PDF>, which provides a useful synopsis of Agile practices, and a series of questions to help non-technical leaders understand whether those practices are being followed.

Product ownership

Taking back ownership of government software projects requires government teams to focus on outcomes, not outputs. This means shifting from some of the traditional Program Management Body of Knowledge practices to a product-oriented mindset.

The word "product" may sound unusual in a government context, but it's an important bit of tech lingo. "Product" is a shorthand for whatever the thing is that's being created: a website, an iOS app, an intranet application, etc. Although the word makes more sense for a business that's selling a literal product, everything else about the concept translates to government perfectly.

The product owner is the key person for any software project, and *must* be a government employee. The product owner works with users, stakeholders, technologists, and the vendor to envision the direction for the product, with an eye toward delivering value to end users as quickly as possible. They iteratively prioritize and define the work for the product team, as part of the Agile process. They measure progress against clear performance indicators, and communicate with stakeholders and the cross-functional team that is building the product.

The product owner doesn't need to be a strong technologist. Instead, they should know the users of the system, the business (for example, Medicaid insurance or DMV services), and policy constraints.

A strong product owner ensures that the vision is clear, the strategy is clear, there is space for teams building the software to learn, and that they are building or buying the right thing to incrementally show value to users. They prioritize ruthlessly to ensure that the product serves user needs, and that activity and attention is focused on the highest-priority needs. They are empowered by their agency to represent stakeholders in making rapid product decisions without the need for many layers of approval. This positioning ensures that the product owner understands everything that the development team is doing and that the needs of government are fully represented.

This is different than typical project management in government IT. The product owner won't have Gantt charts or a detailed five-year plan. But they will have a vision for the outcomes that will be delivered to users, and have a path to executing. Their most important job is to understand what the development team is doing and to make sure it strikes the right balance between the needs of government and the needs of end users.

It's possible for a first-time product owner to learn as they go, but it's better to be trained in advance. There are many sources of Agile and scrum training, some specifically for product owners. These range from YouTube video series to in-person, multi-day classes to become a "Certified Scrum Product Owner." The more important the project, the more formal and rigorous that the product owner's training should be.

DevOps

Historically, the teams building software have been separate from the IT teams that are responsible for operating the software once it's ready for use. A vendor might spend years building new software, and then a government IT team (or a vendor filling that role) might then require many months of work to get that software to function correctly on their servers. This is usually accompanied by frustration and finger-pointing, and can lead to project failures. To address this, government agencies often insist that the vendor building the software also host it indefinitely on the vendor's infrastructure, which has the effect of ruling out most software vendors (who are not in the hosting business), and creating vendor lock-in with its associated high prices. Relying on these old approaches will get you less and cost more than adopting the modern software tools that are standard in the private sector.

The way to address this is with DevOps. This is the practice of coordinating the work of these two groups to automate the work that goes into testing software and moving it to a live server where people can use it—merging software *dev*elopment and system *op*erations. The developers write a series of automated processes for ensuring that the software will function properly in production, over the course of writing the software itself. Developers cannot merely hand their completed work to the system operations team and declare "hey, it works for us"—they are responsible, both practically and contractually, for their code working properly.[5]

Odds are good that most of the software you use every day, whether on your phone or your computer, was written just like this. Under DevOps, testing software quality is automatic, testing software security is automatic, merging multiple developers' work is automatic, and moving completed software to servers is automatic. (The incorporation of security testing in DevOps is sometimes labeled as "DevSecOps.")

[5] For more on DevOps, see the Defense Innovation Board's "Is Your Development Environment Holding You Back? A DIB Guide for the Acquisition Community" <https://media.defense.gov/2018/Oct/09/2002049592/-1/-1/0/DIB_DEVELOPMENT_ENVIRONMENT_2018.10.05.PDF>

Building with loosely coupled parts

Large, complex software projects tend to collapse under the weight of administration. No single developer can understand the entire system that they're contributing to, yet each new member added to a project team increases the complexity of the entire team's interactions. The contributors need to coordinate carefully to avoid conflict between their efforts. No single developer can understand the entire system that they're contributing to, necessitating new supervisory roles like "software architects," with whom developers must check before doing any work. As a team grows, they're forced to spend increasing amounts of time managing the project, and decreasing amounts of time actually doing the work.

To avoid this fate, it's smarter to break large projects into a handful of small, quasi-independent software projects. In this model, each component communicates with other components through simple, modular standards, so that any one piece can be swapped out at any time. Instead of building a monolith that everybody will lament in a few years, you build a little ecosystem, in which each piece can be upgraded and modified easily, as changing needs will demand. Each component is maintained by a single Agile team, which documents the component's application programming interface (API)—the grammatical rules that other components can use to communicate with it. The teams' need to coordinate is minimal, because they can simply follow the API documentation for the other components that they need to interface with.

When each component uses abstracted APIs (think of them as common standards for using that technology), this is known as using "service-oriented architecture" (SOA). This is the same as the concept of "interchangeable parts" that made the industrial revolution possible. Standardized couplings are the underlying concept behind cloud computing, electrical outlets, USB, Legos, trains, and countless other modern products and practices.

Building IT systems using loosely coupled parts, connected by open and available APIs, is the "magic bullet" that allows for flexible, sustainable systems that meet user needs and cost less over time.

Modular contracting

By combining user-centered design, Agile, product ownership, DevOps, and building with loosely coupled parts, it's possible to break up a large, risky contract into a handful of smaller contracts. A contract should be small enough that the agency will have no compunction about giving no further work to a non-performing vendor, replacing them with a new vendor. (See "Procure services, not software" for how this is done.) The rest of the vendors will continue working, so the total loss of velocity will be minimal. A new vendor should have no difficulty taking over for the old one, since the old one was delivering completed, documented, tested software every two weeks. Another benefit is that small contracts may come in under your state's simplified procurement threshold, meaning that agencies can write a request for proposals, publish it, and award a contract, all within 90 days or so.

There are vendor teams that specialize in working as we've described here. As a rule of thumb, an Agile development team of 5–9 people costs between $1–2M/year, depending on their geographic location within the U.S.

This approach will require coordination and buy-in from your procurement teams. Procurement personnel are often accustomed to the traditional approach of outsourcing IT projects: one large procurement based on lengthy RFP documents, asking for lengthy proposals and outdated, waterfall-style certifications and qualifications from vendors. Generally, vendors that use Agile, user-centered methods don't have any idea what "CMMI" or "EVMS" is—such standards are no longer considered best practices for creating flexible and cost-effective software systems. This is a barrier to entry for many of the vendors that might be new to government and don't want to expend all of the resources required to write a proposal.

* * *

Modern software development processes are founded on user-centered design, Agile software development, product ownership, DevOps, building with loosely coupled parts, and modular contracting. By understanding those core concepts, you're in a great position to understand how to budget for software more effectively, and to understand the rest of this handbook.

Best practices for budgeting and overseeing tech projects

Think about risk in a new way

Over the past several decades, government agencies have turned away from using in-house staff, relying on outside vendors to build their mission-critical technology. The decisions to do this were based on trade-offs that seemed like lower-risk options—often driven by limited capacity and promises of cheaper "off the shelf" tools offered by government contractors.

Since then, we've learned the hard way—exhibit A: the Affordable Care Act and Healthcare.gov—that while government can easily outsource the work of creating new technology systems, it cannot outsource the risk of failure. Projects that go wrong reflect back on their agencies, not contractors or software providers.

Government is ultimately accountable for its mission, so agencies need to have control of and responsibility for the projects that support that mission. The problem an IT budget request aims to solve is not a technical problem; it is a problem related to fulfilling the agency's mission, and technology is simply a means to that end.

This doesn't mean agencies need to do all the work in-house; however, it does mean agencies need to set clear expectations about human outcomes and technical standards related to data security, use, interoperability, monitoring, and evaluation.

Technical knowledge is cheap and abundant, but knowing how to run a state agency is a rare and valuable skill. Government must embrace and own its responsibility and risk of failure, recognizing that technology vendors are hired only to help and should be easily replaceable if they don't deliver.

Checklist

- ☐ The project has a dedicated, empowered product owner who is an employee of the mission agency—not a contractor, and not an employee of the state's IT agency
- ☐ Stakeholders recognize that the existing approach (waterfall development) fails the majority of the time, and that moving to Agile development and modular procurement is, in fact, significantly less risky
- ☐ Stakeholders regard outside vendors as interchangeable tools to accomplish a goal, rather than as the "owners" of a project or its outcome

Key questions

- Are there identified and trained government employees (not contractors) that will serve as dedicated and empowered product owners to set direction, prioritize, and oversee the work of the development team?
- Is there a chain of support for this new approach within the agency all the way up to the governor's office, central IT, legal and procurement offices, as well as the legislature? Are any of those stakeholders able to block adoption of this new approach? If so, what is the path to escalating issues, ensuring alignment, and preventing those internal blockers from putting the project at risk?
- How is the agency taking responsibility for leading the project and owning the results, rather than trying to outsource risk to a vendor through the contracting process?

Procure services, not software

Don't think of procuring custom software as buying a *thing*. Instead, think of it as buying a *service*: the service of a team of developers and designers performing work as prioritized by the product owner. This reframing leads to a completely different approach—a much simpler approach—to the RFP and to the contract, and is an important distinction for contracting officers.

Your RFP should describe the overall goal of the work, and should include a first attempt at a product backlog—a list of the work that will be done—put together by the product owner. This should look like a list of user stories—tasks to be performed to address the needs of end users—that the work is likely to address, clearly labeled as indicative of the types of work that's likely to be involved, rather than a fixed scope of work. The RFP should also acknowledge there will be constant change to the work based on shifting priorities and ongoing user research; change is expected, and it's easy to change software when it's built in modern ways.

The RFP should use a Statement of Objectives rather than a Statement of Work—that is, it should state the objectives of the project, rather than the specifics of a product that the vendor should produce. Using a SOO instead of a SOW eliminates "change orders" from vendors, because the scope of work is whatever the team is directed to do. (If an ostensibly "Agile" vendor mentions change orders, that's a red flag.)

To ensure vendors deliver work that meets the needed technical specifications, it is important that the RFP include a Quality Assessment Surveillance Plan (QASP) that is appropriate for Agile development methods, requiring that the software be inspected at the end of each sprint to ensure that it is tested, secure, accessible, documented, and deployed.[6] Meeting this requirement requires regular demonstrations of actual, working software, not memos or descriptions of what a system is supposed to do in the future.

Historically, there has been pressure to only use firm fixed price contracts, on the assumption that this reduces risk. However, if you are in a position to constantly measure software quality, then a time and materials contract—with a ceiling on total spending—allows for more flexibility for the software development team. A time and materials contract also allows for

[6] For an example RFP, see the U.S. Tax Court's 2018 EF-CMS RFQ, which includes a QASP, under the "Deliverables and Performance Standards" section <https://github.com/ustaxcourt/case-management-rfq>.

much easier escape clauses if the direction of the work changes or the vendor team is not producing quality software. If a vendor team's work is inadequate, or their skills prove inappropriate, then no further work need be assigned to that vendor (functionally terminating the contract), and the vendor can be replaced.

Checklist

- ☐ The project has a dedicated, empowered product owner who is an employee of the agency—not a contractor, and not an employee of the state's IT agency—whose has the full-time job of prioritizing user stories for the development team, overseeing the work, and inspecting the deliverables at the end of each sprint
- ☐ An agency contracting officer has embraced this project, and is enthusiastic about procuring software in new ways
- ☐ The RFP will be solely about procuring development services, not about procuring a tangible thing
- ☐ The RFP will require a cross-functional team of designers, user researchers, and developers
- ☐ The RFP will be no more than 20 pages in length
- ☐ A backlog of at least a dozen user stories has been created and added to the RFP as examples of the work to be done
- ☐ A time and materials contract (with a cap) will be used
- ☐ The simplest available procurement vehicle that provides access to the targeted vendors will be used

Key questions

- Is the product owner empowered to rapidly make authoritative decisions on behalf of the agency?
- Is the product owner prepared to spend most of their work hours fulfilling the requirements of this new role?

- Is agency leadership prepared to have product decisions led by identified user needs, based on direct conversations with those users, rather than leadership's personal preferences?
- Does the RFP establish clear requirements about the regular delivery of working code, documentation, testing, and ownership of all work products remaining with the state?

Beware the customized commercial software trap

Commercial off-the-shelf software (COTS) and Software as a Service (SaaS) can be great ways to rapidly procure new software or infrastructure without having to build it from scratch. For example, it makes perfect sense to buy true commodity products like Microsoft Word (COTS) or Google Docs (SaaS) instead of building your own custom word processor.

But for major procurements of specialized, mission-critical technology, be extremely wary of claims that COTS or SaaS will work "out of the box." Vendors will often pitch their "customizable COTS" and SaaS as a magic bullet, promising that it will handle your state's unique regulatory and process requirements. And it might—but likely only after extensive modifications.

Before signing on to those tools, first talk to other state agencies that have used those customized products. Chances are you'll learn that what's being sold as an out of the box solution takes a lot more time and money to customize than you've anticipated.

Instead of mandating any one solution at the budgeting stage, give agencies the space to determine whether to buy or build various pieces of the system. If the budget allocation mandates COTS, then the agency is likely to wind up locked into a highly modified version of a COTS product, cut off from all future upgrades by those modifications without significant expense. Likewise, mandating SaaS is likely to force the agency to cram their needs into a SaaS product like an ill-fitting shoe, while spending a significant amount of additional money on a “software integrator” to connect it to their existing legacy system, leading to the same type of undesirable lock-in.

It may well make sense to use COTS or SaaS as the core of a major new agency system. But the legislature and the agency needs to go into that with eyes wide open, recognizing that they’re not likely to get a completely turnkey COTS or SaaS solution for specialized agency software.

Checklist

- ☐ The budget allocation does not mandate the use of COTS, SaaS, or custom software, but allows the agency to fund a combination of those as they find necessary
- ☐ Vendors’ claims that their COTS or SaaS product will work immediately, without burdensome modification or customization, are independently investigated by talking to other states and agencies that have used those products and gone through the customization and deployment process

Key questions

- How will COTS software updates be made once the product has been customized to meet the agency’s needs? How much further customization will be required to integrate those modifications, and who will pay for those updates?
- What happens if the SaaS vendor goes out of business one day without warning?
- Will the state have no-cost, easy access to its data, data models, and APIs?

Require demos, not memos

Historically, progress in software development projects has been measured by comparing the work that has been done to the schedule of work to be done that was established at the outset. This is done by producing artifacts like Gantt charts and requirements traceability matrices. But this doesn’t work—Agile software development is premised on the idea that this doesn’t work. Modern software development teams have never heard of "CMMI" or "Earned Value Management Systems," and won't bid on work that includes these requirements.

A better philosophy is *demos, not memos.* Instead of measuring progress by looking at purpose-made artifacts, look at the actual work that is being done. Join the reviews that are held at the end of each sprint, where the work done in that sprint is demonstrated to the project team and invited end users. Try out the website. Install the app. Ask for a "burn down chart"—a graph of work that remains to be done and how much time that will take.

An important part of ensuring that progress isn't illusory is for the contract to include a Quality Assurance Surveillance Plan (QASP) that requires, at the end of each sprint, that all work meet specific standards. The QASP describes the method by which the government will determine that the vendor's work is of sufficient quality to accept at the end of each sprint, enabling the vendor to perform those same tests to ensure that there will be no surprises. (See Appendix B for a sample QASP.)

The QASP does not require producing any artifacts explicitly for the purpose of monitoring the work—the way to monitor the work is by *seeing if it actually works.* This is a very different way to monitor the progress of a technology project. It has the added benefit of being a more objective, observable, functional test than requiring subjective or legal interpretations about whether the work satisfies a long series of system requirements.

Checklist

- ☐ An empowered, dedicated government employee will serve as the product owner
- ☐ There will be no planning or reporting requirements that run counter to Agile (i.e., there are no dates by which specific tasks are to be completed and no specifications of exact functionality that will be required—whether in the RFP, the acquisition plan, or legislation)
- ☐ There will be a government-employed software developer who will ensure compliance with the QASP at the end of each sprint

- ☐ People providing oversight, above the level of the government product owner, are willing to primarily receive "reports" in the form of demonstrations of functioning software and burn down charts, combined with a review of user stories that have been completed and those that remain to be completed
- ☐ There is an identified person within the agency who is prepared to provide repeated explanations of progress to each level of oversight, because artifacts of measuring progress on an Agile project are unfamiliar to people accustomed to waterfall projects

Key questions

- Is it feasible to provide the end-to-end support for such a radically different approach to measuring progress, from the agency to the governor's office to the legislature? Is there anybody with the power to dig in their heels and demand a Gantt chart, thus potentially making Agile methodology non-viable?
- Whose job will it be to report progress up and out of the agency, e.g., to a legislative oversight committee?

Hire tech talent in-house

If nobody in the budget office or budget committee has experience with software development, then they are not well-equipped to consider a software development funding request. The same is true of agencies —if nobody in project leadership has experience with software development, then the agency is not well-equipped to lead a software development project successfully. The burden is on the governor's office, legislators, and agency heads to ensure that their respective organizations prioritize hiring people who have this experience.

While it may be tempting to solve this knowledge gap by relying on somebody from the state's central IT department, or by relying on a vendor, ultimately mission agencies must have the knowledge in-house to comprehend what they need, what they should be asking of vendors, and assessing the work done by vendors.

To determine if your budget office or your leadership has the experience to consider software requests or lead software projects, start by asking around. All but the smallest agencies will have technical staff who can join project leadership, although vanishingly few budget offices currently employ software developers.

If you don't currently have the knowledge you need in-house, you'll need to hire someone who does—even if only seasonally or on contract. A developer or designer with experience building modern software, ideally for government, is your best bet. Also, consider authorizing one or more employees to spend some of their training time learning the basics of Agile software development—there are coding "bootcamps" throughout the U.S., including many online-only options.

The personnel cost of bringing in a developer or upskilling your current employees is miniscule in comparison to state spending on technology. And once an employee has monitored an Agile project from start to finish, they'll be better equipped to consider future budget requests for custom software.

Likewise, mission agencies must directly employ enough developers that they can oversee the work being done by vendors. They'll represent the contracting officer, ensuring that vendors' work is of a high quality and that vendors are working on the right things.

Although software is never "done"—you'll always need to adapt to changing technology, policy, regulations, laws, and user needs—there will be a point when you need far fewer developers to continue that work. At that point it becomes especially important to have multiple agency employees who fully grasp the software, who are capable of maintaining it.

For larger projects, you'll need to contract for a development team indefinitely, under the oversight of a government product owner. Under waterfall, this travels under the name of

"Operations and Maintenance," but under Agile, O&M is simply the continuation of user research, design, software development, etc.[7]

Checklist

- ☐ There are one or more budget-office employees with experience developing complex, custom software in an Agile environment who will assist in evaluating custom-software budget requests
- ☐ If there are no budget-office employees with relevant experience, the legislature has a contract with a non-conflicted vendor—one with no other contracts with the state and no ties or partnerships with any COTS products
- ☐ The agency has identified a specific government employee who will be providing technical leadership for the project, along with evidence of their experience developing custom software in an Agile environment

Key questions

- When a vendor delivers code at the end of every sprint, which *specific* government employee will inspect that code to ensure quality?
- If an agency says they need $10 million to complete a specific software project, which budget office employee is equipped to know whether that's an appropriate price? Which *specific* legislative budget committee employee is equipped to know whether that's an appropriate price?
- When the procurement is complete, who will maintain the software? Does the agency employ people who know how to maintain it? Will they be brought into the development process so that they can learn about it as it's built and help ensure it's something they're capable of supporting?

[7] For more about the difference between O&M and continuous Agile development, read "Software maintenance is an anti-pattern" on the 18F blog <https://18f.gsa.gov/2016/02/23/software-maintenance-is-an-anti-pattern/>.

Minimize the cost of change

Your state government will exist longer than any piece of software. And that means one day, the exciting new software system will be the old hated software system.

As good as software may be today, eventually you'll need to switch to a new system—whether that's in whole or in part. And acquiring software as a completed monolith guarantees it will gradually become unable to support an agency's needs.

Technology changes, government policies change, regulations change, laws change, and leadership's priorities change—any project that is planned in great detail up front will be unable to adapt to those changes, and will be at significant risk of failure, significant cost and deadline overruns, or costly "change orders."

So rather than acquiring one giant piece of proprietary software, insist that your vendors default to practices like using open-source software and service-oriented architecture. That way, you can optimize for reducing the cost of updating and changing the system from the beginning.

Checklist

- ☐ Systems, whether cloud-native or being moved to the cloud, will use service-oriented architecture (SOA) that is vendor- and product-agnostic
- ☐ To ensure data portability, files will be stored in open, non-patented formats supported by multiple vendors
- ☐ APIs will use open schemas
- ☐ To avoid product lock-in, open source software will be used instead of commercial software whenever possible
- ☐ Government will own all vendor work products

- [ ] If using COTS components, the vendor will provide a path to leave for a competitor—both contractually and technologically—with a cost-effective way to export all stored data

Key questions

- What is the plan for reducing the time and cost of future updates to the system due to technology, policy, or vendor changes?
- How much will it cost to change the system to reflect needed technology or policy changes?
- Are the APIs open and usable by other vendors?
- Are the data formats standardized, open, and usable by other vendors?
- Keeping a software system up-to-date will take regular, on-going work—what is the plan to do that?

Measure success based on iterative outcomes, not project milestones

Value shouldn't come at the end of a project—it must be provided to end users within no more than six months of the contract being awarded, and constantly from there on out. At the end of the *first* sprint, working code must be delivered to the agency for its review, and that must continue with every subsequent sprint. End users should evaluate work at the end of each sprint, regardless of whether the work has yet been deployed for daily use.

Don't measure progress in "story points," lines of code written, person-hours of work, etc. The only measure of success that matters is what value has been delivered to end users. This is best assessed by attending twice-monthly sprint reviews and talking to both the scrum master and the government product owner.

Checklist

- ☐ The vendor team will use Agile
- ☐ The vendor will be required to deploy functioning software into a government-owned environment at the end of each sprint
- ☐ The project team will interview and test their work with end users routinely, both to inform planned work and to determine whether the work already done is correct
- ☐ The RFP will have no mention of a detailed project schedule, and there will be no mention of Gantt charts or Independent Validation and Verification (IV&V) contracts
- ☐ A legislative staffer will be assigned to provide oversight of the project, and will coordinate with project leadership to monitor progress by periodically attending sprint reviews

Key questions

- Can the requesting agency deliver value to end users within six months? What, specifically, is that value?
- Is the agency prepared for the vendor to continuously interview and test their work with actual end users of the software—perhaps including agency employees?

Limit total spending

The greater the amount of money spent on a software project, the greater the odds of failure. As a general rule, plan to spend no more than $10 million on an entire project.[8] (There are rare exceptions for extraordinarily complex systems like unemployment insurance, Medicaid Eligibility & Enrollment, and Medicaid Management Information Systems.)

[8] In The Standish Group's 2015 CHAOS Report, based on a survey of over 25,000 software projects, they found that software projects that cost more than $10 million succeed only 6% of the time. (Their more recent reports are just as grim, but require a paid membership to access, making them poor citations.) Outcomes improve substantially as the dollar value is reduced, peaking at a 61% success rate for projects under $1 million <https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf>.

Checklist

- ☐ The requesting agency understands they're not being given a small percentage of the resources they believe they need—instead, they're being given an entirely new process to procure software, as well as adequate funding under that model

Key questions

- If the project "requires" $20 million in funding, what value can be delivered to end users with $10 million? Or $2 million? (If the answer is "none," then this is a project doomed to fail)
- If this spending is matched by federal dollars—especially at a highly-leveraged rate, like the 9:1 match provided by the Centers for Medicare & Medicaid Services for Medicaid Management Information Systems—is anybody going to get in trouble for leaving money on the table?
- Is there somebody whose performance is measured by how much grant funding they raise and who has an incentive to demand that $100 million be spent, instead of $10 million?

Limit contract sizes

Using a single vendor over a long period of time, or for a large number of teams, may feel more comfortable, but it inevitably leads to vendor lock-in. Breaking up projects into several small contracts incentivizes vendors to build a sustainable software ecosystem, instead of a monolith, and makes each contract small enough that the odds of success increase markedly.[9]

[9] In The Standish Group's 2015 CHAOS Report, based on a survey of over 25,000 software projects, they found that software projects' outcomes get worse as more money is spent <https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf>. Limiting the spending on each contract segments the project into smaller components, making each component—and the entire project—more likely to succeed.

Require that no more than $3 million be spent on any single contract annually, and that no contract last for more than three years, including option periods. That way, you'll get no more than two development teams from a single vendor. If the project needs more development teams, obtain them from another vendor and have them work separately. Limit the RFP, too, keeping it below 20 pages; don't spend any longer than 60 days writing it.

In addition to preventing lock-in, there's another benefit to using smaller contracts: they're less likely to be protested, because the dollar value doesn't justify the trouble and legal costs. If you're respectful and transparent with vendors, and don't require hundreds of pages of proposals, they will likely want to do business with your agencies in the future.

As the number of people who work on a project increases, so does the amount of time that all of those people have to spend coordinating with each other. The solution to this is to have them work in parallel, which is possible when building with loosely coupled parts. Having more than one vendor team working on your project also provides you with more competitive options if you need to change vendors.

Checklist

- ☐ If the project will require multiple contracts, the scope of the first contract has been identified, and there is a general idea of what some other contracts may be comprised of
- ☐ If there will be more than one development team, service-oriented architecture (SOA) will be employed
- ☐ When possible, contracts will be sized within the simplified procurement threshold so they can be awarded quickly and easily
- ☐ The identified first project has relatively low technical complexity, low political risk, and high end-user value, so that teams can start practicing working this way while experimenting and learning in a relatively low-risk environment

Key questions

- Have the relevant contracting officers read this handbook?
- Do the contracting officers understand that they're not being asked to do all of the work that goes into a $50 million contract? Do they understand that $2 million contracts are far easier to award and that, under Agile, they'll also be much easier to manage?

Fund systems, not monoliths

Don't replace the old legacy system with a new legacy system. Insist on loosely coupled systems that are built incrementally. That way, they'll never need to be replaced wholesale—they'll just replace individual components as the need arises.

Checklist

- ☐ Each contract will be written to deliver value to end users—they're not for "maintaining servers" or "setting up a database," but for "adding a web-based permit application system" or "simplifying the intake process"
- ☐ There will be no single "enterprise architect," because the architecture will emerge iteratively throughout the Agile process
- ☐ If the project is large enough that it will include multiple scrum teams working simultaneously, there is no expectation that all members of all teams will ever be in meetings together
- ☐ The RFP will specify the use of service-oriented architecture for each component

Key questions

- Is there a single point of failure that can bring the whole system down? (If so, that's probably a monolith, not a system.)

- If one vendor’s contract needs to be terminated for non-performance, can the others continue to work without interruption?

Expand your vendor pool

Your existing vendors are unlikely to employ the modern software development practices outlined in this handbook—they were hired for their legacy practices back when that was what you needed. To find vendors who meet your new needs, you’ll likely need to identify and attract new companies that use modern software development practices.

If it is important to get bids from in-state vendors, then know that odds are good that there are many qualified small businesses that can deliver Agile development programs in your state.[10] However, if you want to drive down the price of bids, then it’s important to consider working with remote or distributed (rather than on-site) vendor teams.

In places like California, Washington, New York, Virginia, and Maryland, the cost of an Agile team can be easily twice the cost of a team in the Midwest or the northern plains—a difference of over $1 million each year with no difference in quality. That’s a price point at which it’s worth rethinking how important it is that the team be local, or even in-state.[11] Encouraging distributed vendor teams also gives you access to a wider talent pool, so it’s smart to embrace distributed working best practices to engage with development teams[12] and only require on-site visits when necessary (such as for user research).

[10] Alaska’s Department of Health & Social Services faced this challenge in 2017, and their Contracts and Procurement Manager wrote about the process that they used to attract small, Agile, Alaskan vendors in “How Alaska is using transparency to attract modern software vendors” <https://18f.gsa.gov/2017/09/12/how-alaska-is-using-transparency/>.
[11] The Bureau of Labor Statistics provides state-level wage information for software developers, which shows that the difference between the most expensive developers (Washington state) and the least expensive (Puerto Rico) is a 150% wage gap. Even within states there can be tremendous variation in labor costs between urban areas and rural areas. As a result, insisting that vendor teams work on-site can double the cost of software.
[12] See “18F’s best practices for making distributed teams work” for specifics <https://18f.gsa.gov/2015/10/15/best-practices-for-distributed-teams/>.

How do you find those qualified small businesses? There are a number of cities and states that have created a pool of Agile software vendors. For example, California's Department of Technology has a vendor pool that adds new companies on a rolling basis. Seek out and draw from these vendor pools, and include some of these companies in future RFP processes. Also, ask around among colleagues in other state agencies to see if they can recommend any vendors to include. Finally, try thinking like a software developer looking for a job, and use sites like LinkedIn, Glassdoor, Monster, and ZipRecruiter to identify Agile vendors in your state. This entire process only takes a few hours.

Although the procurement team will be tempted to seek out vendors who have previously built a near-identical system, that's both unnecessary and limits the vendor pool to just a few big, international companies. Instead, they should widen their scope to look for vendors that have built something analogous. A vendor that has built a website to book rental cars can build a website to apply for backcountry camping permits. A lead developer who has built a database to track the positions of comets can build a database to track state-owned vehicles. By seeking relevant expertise along this axis, the procurement team will find plenty of developers who can get the job done.

Checklist

- ☐ The RFP will be streamlined (no more than 20 pages), and comprehensible by software developers who do not normally work with government
- ☐ The acquisition plan includes reaching out to small vendors to encourage them to bid
- ☐ The RFP will not be hidden on a registration-required procurement website, but published openly on the web so the vendor community can share it
- ☐ The RFP will require that vendors name their key personnel in their proposals—no more than three people—such as the lead developer or the lead designer
- ☐ The acquisition plan includes interviewing the finalists about their proposed approach, questioning the named key personnel, *not* the vendor's sales staff
- ☐ Vendor employees will not be required to work on-site at a government facility

- [ ] Vendor teams and the government product owner will be permitted to use a desktop-based video call service (e.g., Zoom or Google Hangouts), a real-time collaboration tool (e.g., Slack or Teams), and a public version-control system (e.g., GitHub or GitLab)

Key questions

- Are there any benefits—political or otherwise—to awarding contracts to in-state vendors, or even requirements to do so? Might that limit the degree to which you can expand your vendor pool?
- Is $1 million per year savings for each scrum team sufficient to overcome any objections to remote teams?
- Has lightweight market research been done to know what vendors will be targeted with the RFP, rather than only issuing an RFI and hoping for the best?

Share your software

An agency's software is likely to be useful, in whole or in part, to other agencies within the state, to local and regional governments within the state, or to similar agencies in other states. Additionally, in many states software created as a work of government is inherently in the public domain, which means an open-records request is all that's necessary for software to become public.

If the software is published openly, vendors' employees will be eager to work on it—it becomes a rare case of work that they can add to their portfolio for future jobs or share with friends, which helps to ensure that you're getting their best work. Also, additional RFPs issued for the project can direct vendors to the code that's already been written, allowing them to see exactly what they'll be working on or interfacing with.

Checklist

- ☐ The RFP will require that software source code be written and maintained in public on a social-coding platform (e.g., GitHub or GitLab), from day one
- ☐ The RFP will require that software be explicitly dedicated to the public domain or published under an OSI-approved open source license <https://opensource.org/licenses>
- ☐ The RFP will use best security practices by requiring that software be strictly separated from data and secrets (e.g., passwords), with automated testing to make sure that separation is maintained
- ☐ The RFP will require that software be documented sufficiently well that a developer with no connection to the project can use it to run their own copy of the software

Key questions

- Will the state or agency security office bristle at the prospect of publishing open-source software and block deployment of the software?
- Are there other agencies in the state or elsewhere around the country who are likely to benefit from this software? Can they be consulted prior to and during the development process?
- Will the agency's office of general counsel (or its equivalent) object to publishing software in the public domain or under an OSI-approved open source license?

Budget for software as an operational expense

Unlike bridges and other capital infrastructure projects, custom software is never "done," so it's important to plan for it to be modified continuously. That way it can serve today's agency needs, not yesterday's.

For small systems, this may require adding one or fewer FTEs to the agency's staff of software developers. For large, flagship systems, this may require procuring a team of developers to continually develop and maintain the software.

Software maintenance is sometimes budgeted for as if it is a different activity than initially building software, but that is a mistake. Maintaining software should mean simply continuing to modify it in response to identified user needs, which change continuously along with laws, regulations, policies, best practices, and technology. This requires the same skill sets, methodology, and tasks as building a system in the first place. A proposal to transition software development into an "operations and maintenance" ("O&M") phase should be seen as a red flag,

Rule of thumb: a "scrum team" of 5–9 developers costs $1–2 million per year, depending on the cost of living in the area where the developers reside. Funding can be ramped up over the course of several budget cycles, as the requesting agency demonstrates that they're successfully reducing risk, controlling costs, and delivering iteratively to end users.

Ultimately, this can provide agencies with a predictable source of funding for software projects—replacing unpredictable capital expenditures—while simultaneously providing the legislature with a predictable annual cost for all agency software projects.

Checklist

- ☐ The agency recognizes that software must be improved continuously as long as it is in use, because "maintenance" is functionally the same as building software in the first place
- ☐ The agency plans to procure Agile development services
- ☐ You have talked with the requesting agency to determine if they would prefer to receive funding over years, as a predictable stream of operational funding, instead of as a lump sum

- ☐ This approach has been coordinated with the governor's office, the budget office, agencies, and the state IT agency—this is likely a radical change that will require trust and cooperation between all parties
- ☐ If an agency's request is at a high risk of failure, you will allocate only a few million dollars in the first year, increasing funding as the project delivers value

Key questions

- Is the requested funding going to be spent within a single budget period?
- Perhaps $50 million is being requested, but what value can be delivered to end users with $2 million? And the next $2 million? And so on?
- If this project is being funded using federal dollars, is the federal agency amenable to taking an operational approach to funding?

Ask technical questions of agencies

Budget requests for custom software often feature non-technical people making a technical proposal to other non-technical people. This process doesn't lend itself to asking key questions, such as many of those found throughout this handbook. It is important to ask all of those difficult technical questions, and to insist on getting the right answers (see Appendix A for sample questions and answers).

It is no kindness to fund a project that is going to fail. If the agency doesn't know exactly what they want to buy, they're not going to get it.

Checklist

- ☐ The agency will rely on the U.S. Digital Service's Digital Services Playbook <https://playbook.cio.gov/>

- ☐ If building a website, the agency will direct the vendor to use the U.S. Web Design System <https://designsystem.digital.gov/>
- ☐ The agency will adhere to the Defense Innovation Board's "Ten Commandments of Software" <https://media.defense.gov/2018/Apr/22/2001906836/-1/-1/0/DEFENSEINNOVATIONBOARD_TEN_COMMANDMENTS_OF_SOFTWARE_2018.04.20.PDF>
- ☐ The agency has read this handbook
- ☐ The rules and questions found in the Defense Innovation Board's "Detecting Agile BS" guide <https://media.defense.gov/2018/Oct/09/2002049591/-1/-1/0/DIB_DETECTING_AGILE_BS_2018.10.05.PDF> have been applied to and asked of the agency, and their answers are satisfactory

Key questions

- What exactly does the agency want to buy? Why? Who will benefit?
- Which parts of the system will be custom? Which will be actual (not customized) COTS? How much will those updates cost? What will be done when a commercial component ends production—e.g., if the database company goes out of business?
- Who are the end users of your system? Have you talked to them? What do *they* want?
- Are you prepared for when changes need to be made?
- How much will it cost to move to a new system?
- What are you doing to avoid paying expensive change fees in the future?

Appendix A: Questions to ask

When you are considering a budget request for a custom software project, it will be difficult to consult this entire handbook to find the right questions to ask. Here are some basic, open-ended questions that you can ask to determine if a project is set up for success.

What are the goals of the project? What outcomes are prioritized?

Wrong answer:	Anything technical in nature, instead of about improving the user experience.
Right answer:	One or more specific user needs are named.

What is the user need that this project will address?

Wrong answer:	Anything that doesn't name clear needs of end users identified via user research.
Right answer:	The agency has determined specific needs based on interviews with end users, and can name several of those needs specifically.

If the selected vendor doesn't perform adequately, how difficult will it be to terminate the contract? How long will it take to replace them with another vendor? How much do you think that will cost?

Wrong answer:	"We would be very reluctant to terminate the contract. It would take months or years to replace them with a new vendor. Significant staff time would be required to do that, and it would set our project back by many

months. Once we have a system, we'd have to start all over if we decide to change vendors."

Right answer: "It will be a time and materials contract, so we could stop assigning work to the vendor at any time, and that would be the functional end of the contract. We could reissue the RFP and have a new vendor onboarded within six weeks. It would require a small amount of staff time, and it would set the project back only by those six weeks."

Will the RFP include requirements for how the system will operate? If so, how many requirements are included?

Wrong answer: "We've spent the past year reviewing our business requirements, and we've written hundreds of requirements to include in the RFP, to ensure that we get exactly what we need."

Right answer: "We're more focused on the outcomes we want from the new system. We've developed a backlog of user stories to help guide the team's work, rather than producing a detailed list of technical requirements."

How long do you expect the RFP will be?

Wrong answer: "We've developed several hundred pages of system requirements along with 50 more pages of standard terms and conditions."

Right answer: "Less than 20 pages, and we expect to keep this under the state's simplified procurement threshold, to make it easier, cheaper, and faster for new vendors to bid on the project."

Do you anticipate issuing a fixed price contract, or a time and materials contract?

Wrong answer:	“Fixed price, because it’s the best way to control vendor costs.”
Right answer:	“Time and materials, because it’s the best way to retain the flexibility we need to respond to user needs, manage unforeseen technical challenges, and ensure vendors that aren’t delivering what we need can be changed without putting the project at risk.”

What value will be delivered to the users within six months?

Wrong answer:	“None—it won’t be ready by then. We plan to show it to users when everything is finished.”
Right answer:	Specific examples are named.

Who will be the product owner?

Wrong answer:	“What is a ‘product owner’?”
Right answer:	A specific person is named, or they’re training in-house staff to take on this role.

What software development process will be used?

Wrong answer:	“Waterfall,” or any answer that indicates a lack of comprehension.
Right answer:	“Agile” and “Scrum” are both acceptable answers.

On the team that prepared this request, who has experience developing software?

Wrong answer:	"Nobody."
Right answer:	A specific person is named.

How often will work be deployed into production?

Wrong answer:	"When it's done."
Right answer:	"At the end of each sprint."

Will the project automate testing? Integration? Deployment? Security tests?

Wrong answer:	"We're looking into that."
Right answer:	"Yes, from day one."

How much will change orders cost?

Wrong answer:	Any response that foresees change orders of any kind.
Right answer:	"We expect the system to change constantly in response to new user needs, new technology and new policy so that's why we're using a time and materials contract and an Agile development approach to lower the cost of responding to these changes."

How will you know if the project is on track and that contractors are delivering as promised?

Wrong answer:	"We're contracting with an independent verification and validation (IV&V) expert to provide us with monthly reports on the project's status."
Right answer:	"Vendors will provide frequent demonstrations of working software that reflect our priorities, meet the technical standards of the QASP, and provide value to end users. If these standards are not met, and value to end users isn't shown within six months, they'll be terminated."

Who will own the software?

Wrong answer:	"The vendor."
Right answer:	"The state" or "it will be committed to the public domain."

Appendix B: Sample Quality Assessment Surveillance Plan (QASP)

Per the "Require demos, not memos" best practice, here is a sample QASP, which should be incorporated into Agile software RFPs.

Quality indicator for	Performance standard	Acceptable quality level	Assessment method
Tested code	Code delivered under the order must have substantial test code coverage and a clean code base	Minimum of 90% test coverage of all code	Automated testing
Properly styled code	Meets acceptable quality level	0 linting errors and 0 warnings	Styling standards and linters
Accessibility	Web Content Accessibility Guidelines 2.2—"AA" standards	0 errors reported using an automated scanner, and 0 errors reported in manual testing	Automated and manual testing

Deployed code	Code must successfully build and deploy into a staging environment	Successful build with a single command	Live demonstration
Documented code	All dependencies are listed and the licenses are documented Major functionality in the software/source code is documented in plain language Individual methods are documented in-line using comments that permit the use of documentation generation tools such as JSDoc A system diagram is provided	Vendor provides above documentation	Manual review
Security	Open Web Application Security Project (OWASP) Application Security Verification Standard 4.0.3	Code submitted must be free of medium- and high-level static and dynamic security vulnerabilities	Evidence of automated testing per OWASP

User research	Usability testing and other user research methods are conducted at regular intervals throughout the development process (not just at the beginning or end)	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 vendor's research plan	Demonstrated evidence of user research best practices