State of Vermont IT Funding

Reducing Risk of Major Software Projects

Waldo Jaquith, U.S. Digital Response
April 17, 2025

Good morning. My name is Waldo Jaquith. Thank you for inviting me here today. I'm sorry I can't be there in person—I live in Virginia—so I appreciate being able to testify remotely.

I'm with U.S. Digital Response, a nonprofit, nonpartisan, grant-funded organization that works alongside government at all levels to ensure they have the capacity to meet the public's needs. My background is as a software developer, but for the past decade, my work has been on the procurement, budgeting, and oversight of major government software projects, to stop them from failing. I've previously worked in appointed and senior positions at the White House, the U.S. Treasury, and the U.S. General Services Administration.

I'm here to talk to you today about how to reduce the risk of major IT projects—specifically, of custom software projects—through changing the funding and oversight models for those projects.

I'm going to start by telling a story of a very normal failed state software project. Then I'm going to explain how the private sector develops software in 2025—because I want to empower you with that knowledge. And finally, I'm going to leave you with some recommended actions that will allow you to take control of major custom software projects to ensure that they can succeed.

Case Study

Let's start with the case study. This story is not special—there are hundreds very much like it. This story happens to have Rhode Island and Deloitte as the two main characters. Please understand, I'm not trying to single either of them out—I'm just telling you a story of something that happened.

This is the story of Rhode Island attempting to put together a unified health infrastructure program. They wanted one application point for state residents to be able to get a bunch of benefit programs like Medicaid, food stamps, child care, TANF—that sort of thing. Instead of applying over and over again, just let folks apply once and qualify for all of them.

They started with an executive order a few governors ago. That turned into a contract with Deloitte in 2013. It was a $100 million contract, and the work was due to be delivered two years later, in July 2015.

This was a major initiative for Rhode Island, and a major initiative for Gina Raimondo, who became governor in early 2015. Three hundred thousand Rhode Island residents—which is a lot on the scale of Rhode Island—would rely on this UHIP program in some way.

The years went by. There were lots of change orders, lots of contract modifications, and that brought the price tag up from $100 million to $364 million. It launched not in July of 2015, but in September of 2016.

Governor Raimondo had heard horror stories about similar projects in other states, so she asked Deloitte's leadership for their personal assurance that things were all set, that things were going to go well. And she was given a "stoplight chart," all full of green lights, indicating everything was fine. "We've dealt with all the risks."

In planning for a successful switchover to this new computerized system, the state Department of Human Services laid off over 60 state employees—particularly field staff and veteran managers. A press release from the governor's Department of Administration declared the layoffs were because of their about-to-be-released modern digital system, which would require different staffing.

Then they had what's known as a "big-bang deployment"—they shut down the old systems on a Friday and got the new ones running over the weekend to be ready to go on Monday morning.

It goes the way that big-bang deployments always go: it was an absolute debacle. Medicaid recipients were cut off. No SNAP benefits were awarded. Some of those laid-off workers couldn't claim their own benefits because they weren't there to answer their own phone calls.

This led to enormous problems in the lives of hundreds of thousands of Rhode Island residents. I'm not going to belabor this point because it's not what I'm here to talk to you about, but the failure of this software caused huge, life-altering problems for vast numbers of people. I'm sure an quantifiable number of people died from lack of access to essential benefits.

The governor appeared to stand by for five months while Deloitte offered excuses and software patches, none of which made meaningful progress. This was a political humiliation for Raimondo. It was not Deloitte's name in the headlines for long—soon, it was the governor's name.

Blaming a vendor will carry you through a news cycle or two, but it's not going to cover you for long. This was a huge failure—not of software, but of government. Rhode Island outsourced their ability to accomplish a core part of their mission to Deloitte. And this was load-bearing software—without the software, these agencies could not do their job.

Deloitte was forced to apologize to the legislature. I assume the most important person who couldn't get out of it was marched in front of a committee meeting much like this one. And as punishment, they were given a new $99 million contract.

The punishment for blowing a $100 million—turned into a $364 million—contract is that you get another $100 million. And that contract can be extended through 2030.

Throughout this process—there was a really great independent report produced for the state of Rhode Island that I'm drawing from—everybody followed the normal policies and procedures. This all went very normally. Everybody followed the rules.

But nobody was on the hook for the outcome.

The theory was that the vendor was on the hook for the outcome, but there's plainly no sense in which that was true. Their punishment for a bad outcome was being given an eyeball-popping additional sum. Their incentive is to *not* succeed.

In government, nobody gets fired for a hundred-million-dollar failure because government oversight is set up around process adherence, not the quality of outcomes. We pretend outcomes are the vendor's concern, but why should the vendor be concerned with the outcome if they're just going to be given more money when it doesn't go well?

The independent assessment delivered to Rhode Island's governor in 2017 found two major problems. First, the state "did not have the capacity to recognize Deloitte's shortcomings." And second, Deloitte's design process "did not adequately account for the input of state workers and other end users of the system."

In other words, Rhode Island was being sold a false bill of goods, and they couldn't grasp that. And Deloitte delivered software that in no way accounted for the actual needs of the intended users.

These are very normal reasons why these projects fail.

There's a coda to the story.

Thanks to that $99 million contract, Deloitte has run this system all along. In December, UHIP—since renamed Rhode Island Bridges—was hacked. All the data was stolen and held ransom. This is the private health data of over half of the residents of the state of Rhode Island. Rhode Island ordered Deloitte to get the attackers out of the system, but after a few days, it emerged that Deloitte was unable to do so. So Rhode Island had to order them to shut down the state's benefits system. The Department of Human Services fell back to working on paper.

At first, Deloitte blamed Rhode Island for this attack, but that story didn't hold up, since Deloitte had built the system and operated it for the better part of a decade. Blaming Rhode Island was nonsense.

As of two weeks, the system was finally fully operational again, according to the Human Services director, testifying before the Rhode Island House Committee on Finance in a four-hour hearing. She also testified that there's been a decrease in public use of the system, because people don't trust it. Six hundred fifty thousand Rhode Islanders will be dealing with the effects of identity theft for years to come.

Again, I don't tell this story to bash Rhode Island or Deloitte. Every state has multiple stories like this, involving any of a dozen usual-suspect vendors who just get most of the contracts. Neither Rhode Island nor Deloitte is unusually bad at this. This is all normal. Horror stories like this can happen in small states like Rhode Island and Vermont as easily as they can happen in large states like New York or California.

I want to give you a few statistics to help you understand how normal this is.

According to research by the Standish Group—the major consultancy in this space—when you spend more than $6 million on a custom software project, only 11% of those projects are successful. "Successful" means they're within 20% overrun of cost, 20% overrun of schedule, and a 20% shortfall in the required performance.

The more you spend, the worse your odds. If you spend $20 million, you have a 2% chance of success. So when you see requests for a $10, $20, $50 million project, please understand—that's most likely to fail. I'm just looking at the stats.

I love this study the Department of Defense did. It spends a huge amount of money on custom software. They looked at $35 billion worth of spending over many years and found that 46%—almost half—met the requirements of the contract, but did not address the needs of the people who needed that software.

So those projects might have been “successes,” even if delivered within cost, schedule, and performance. But half of them aren’t even doing things that people need, because the requirements aren’t capturing the needs of end users.

Another study from IEEE’s journal, as part of the NASA Goddard Software Engineering Workshop, looked at 400 software projects. All the code had been written to power them, but only 10% of that was ever actually put in a production environment where people used it. And of all of the code written, only 2% was ever actually used by anybody.

We’re spending a huge amount of money on software that doesn’t even do what people need it to do.

But wait—it gets worse.

A great study conducted by project management researcher Bent Flyvbjerg, a Danish economic geographer, looked at data from the U.S. Office of Management and Budget for years. He looked at thousands of projects—$56 billion worth—and found that the average government IT project costs over three times the originally estimated price. And that’s the average. Half of them are even more.

It’s really grim. The status quo is really grim.

What I want to tell you here today is that it doesn’t have to be like this. There’s a much better approach available to you. This approach fails at a much lower rate, and when it fails, it’s at a way smaller dollar value.

I care about this not because I’m interested in project management—I’m not. I care because a huge amount of the mission of government is intermediated by technology. When we outsource the technology, we are often outsourcing the mission.

You care about your mission. But it’s not the vendor’s job to care about your mission. They care about making money. And that’s fine—that’s good and healthy. But we need to get those to overlap better.

Modern Software Development Practices

Let’s talk about how you can reduce risk to the point where it ceases to be a major concern, by restoring control over projects to the government.

There are five things agencies need to have in place to take control of software projects:

1. Agile software development
2. Product ownership
3. User-centered design
4. Building systems out of loosely coupled parts
5. A procurement approach that allows for all of this

These are all very normal, standard, accepted practices for building software in the private sector. That memo just hasn't made its way into government adequately. It's gotten a lot better in the past 5 or 10 years, but we've got a long way to go. I want you to know the signs to look for to know whether a project is set up to succeed.

1. Agile Software Development

This is a way to build software that ensures it will do what it needs to do. It addresses the DoD problem—spending a lot of money on stuff that doesn't actually solve any problems.

Let me start by telling you what Agile is not. Agile is not what's derisively known as "waterfall" software development. That's how software is mostly built in government right now.

Here's how that usually works: some consultants come up with a bunch of requirements. Those go into an RFP, and a contract is awarded. The requirements are handed off to designers, who design screens. They give those screens to engineers, who write the code. Then there's some verification step to make sure all the contract requirements have been satisfied—usually with a requirements traceability matrix. Then it goes into a maintenance phase.

Those requirements often come straight from legislation. If the legislation is particularly prescriptive, it'll just be quoted verbatim in the RFP.

This is how software was developed 50 years ago—and still mostly 30 years ago in the private sector. But it's not used anymore in the private sector. It's the standard approach in government, and it's a big part of why government software projects fail.

The alternative is Agile software development. This is used by 89% of professional software developers in the U.S. It is super normal. The concept of Agile is not that special.

Instead of planning a multi-year project in great detail up front, you define the objective. Maybe you're modernizing a system. Maybe you're providing web access where none exists. Then the

vendor team begins by building something small that makes progress toward that objective—in just two weeks.

After two weeks, government inspects the work. Real users verify that it's making their lives better. From what they learn, the vendor plans what to do for the next two weeks. And so on. It's iterative. Every two weeks, there's progress.

You don't define a million requirements in a contract. You define requirements every two weeks. That's how the private sector works, and we need to bring that to government. This works well for a brand-new system, but it works *even better* to modernize or replace an existing system. Instead of waiting years for results, the public benefits almost immediately.

Here's the takeaway: Agile gives you extraordinarily tight contract oversight of vendor activities and ensures you're getting your money's worth. You see working software every two weeks.

Contrast that with the waterfall approach, where the vendor disappears for months—or years—working in the dark. No benefits are delivered until the end. Then you get there and find out the work wasn't done, and they want more money. That's not working for anyone.

2. Product Ownership

Government software systems are not "products," strictly speaking, but treating them as if they are gives us a useful framework. Just like thinking of taxpayers as "customers" can help, too.

The key role here is the product owner: one person who considers the needs of users, the demands of stakeholders (like you), the limitations and possibilities of technology, and any business or legislative requirements.

Their job is to orchestrate all of that so the software does what it needs to do. Every major system needs a full-time product owner. If an agency just has a governance board or a project manager, that's not set up for success.

Here's your takeaway: the product owner is the fulcrum on which the success of the project hinges. They understand everything the vendor is doing, in exhausting detail. Oversight is vastly easier when there's one person with all the answers—or who knows where to get them.

3. User-Centered Design

Also called “human-centered design.” As with everything else here, this is just how the software industry works.

I love this quote from a relatively unknown Facebook employee: “Design is deciding how a thing should be.” It’s not just choosing colors or making it pretty—it’s deciding everything about how the product works.

User-centered design means we talk to real people who will use the software. We observe their work. We learn what they need. Then we design solutions, verify that they meet those needs, and repeat. Constant improvement.

If you’re not designing for and with the people using your product, you’re designing for imaginary people. And you’ll get imaginary results.

The best software vendors will employ nearly as many user researchers as coders—and they’ll conduct user research throughout the project.

Without it, you’re flying blind.

4. Building Systems Out of Loosely Coupled Parts

Modern software systems are not monoliths that you have to replace all at once. They’re ecosystems made up of small, independent components—each performing a unique function.

Here’s a metaphor for why that matters: containerized shipping.

Before containers, it took weeks to load ships—barrels, bags, boxes of all shapes and sizes. In the 1950s, a few people designed a standard shipping container. They gave away the patents so anyone could use it. The result? The cost of shipping fell by 97%. Today, cargo can be shipped across the Pacific for sixteen cents a ton.

What matters isn’t what’s in the container—it’s that everything fits together. You can swap one container for another, stack them, move them. It’s all standard.

In software, we call these interoperability standards, or APIs—application programming interfaces. APIs are how software talks to other software.

When your email program checks your calendar to see if you’re free for a meeting, that’s just one API talking to another. It’s a standardized way of connecting different parts.

Your IT staff could swap out your mail server from Outlook to Gmail and everything would keep working. That's because email has a common API standard.

Modern software systems are just like that: a bunch of interoperable pieces. You don't have to care what's inside each one. What matters is how they connect.

Why does this matter?

Because whatever fancy new system you're building today, it will eventually become the hated old system. Every legacy system started out as something shiny and new.

So we want software systems that can be continuously improved, one piece at a time. If one vendor isn't doing a good job, that's okay—just replace them. You don't need to replace the whole system.

This also means multiple vendors can work at once, each on different parts. You don't need to award one giant contract.

5. The Procurement Approach That Makes All of This Possible

This is known in federal procurement—and in many states—as modular contracting. You break up big, complicated procurements into a series of smaller ones, and you implement the system in pieces. The Federal Acquisition Regulation recommends this specifically for complex IT projects.

There are two key principles when procuring big software systems:

1. Don't put more than $10 million in one contract. The more you spend, the higher the odds of failure. Break it into smaller pieces. Don't put all your eggs in one basket.
2. Keep contracts short. These five- or ten-year contracts are just trouble. The longer they go, the greater the risk of failure and the harder it is to change vendors. Don't let contracts last more than three years.

Now, here's where strategy comes in.

The status quo in many states goes like this: spend years preparing an RFP, gathering tons of requirements—often without talking to any end users. Finally, get a vendor under contract, spend a fortune, and then…wait. Nothing is delivered until the very end. And even then, it probably doesn't work.

This approach fails at a really high rate. It's expensive. It's overly prescriptive—telling vendors exactly what to build, leaving no room for innovation. It's time-consuming. And the public gets zero benefit until the very end. Knowledge stays with the vendor the whole time. Agencies get locked in. If they want to switch vendors, they can't.

Here's the Agile approach. Acknowledge that things will change—policy, law, leadership, technology, user needs. Use small contracts that take months—not years—to award. Let multiple vendors work at once. Deliver value to end users continuously. Let users verify: "Yes, this is good. This is making my life better."

It's cheaper, faster, more flexible. It encourages innovation. It keeps risk low. And most important, government doesn't get held hostage by one vendor.

What You Can Do: Legislative and Executive Actions

Let me talk about what you can do—what legislative actions you can take to improve on the status quo.

We've talked about how software is built in 2025. But government software procurement in most states is still stuck in a 1990s paradigm, while the rest of the world has moved on. Here are the levers you can pull to reduce the risk of failure.

First, let me explain how a state can all but guarantee its software projects will fail. If that's your goal, do the following:

- Make sure agencies never talk to end users to find out what they need. Instead, pay consultants to write requirements based on guesswork, then pay a vendor to build those guesses.
- Load up contracts with thousands of detailed requirements, and create staffing structures to enforce those—regardless of whether they actually help anyone.
- Sign absolutely enormous contracts. Gather every egg you can find and put them all in one basket.
- Conduct oversight only via reports—never look at the software. Base your understanding on reports prepared by independent vendors, based on reports they got from the primary vendor.
- And finally, if an agency figures out early that a project is going off the rails and cancels it to save money—investigate them, threaten them, make it clear that saving taxpayer dollars gets you punished, not praised.

If Vermont does all that, you can be sure your major software projects will go really badly.

Now, on a more cheerful note, here are some specific executive actions the administration could take—and that you could encourage:

1. Demos, not memos. Any large software project should demonstrate working software every two weeks. These demos should feature improvements made in the past two weeks. They should invite the Agency of Digital Services to each demo. That's vastly better than any written report. Stoplight charts are always green—until they're suddenly red at the end. Don't rely on reports; inspect the work.
2. No custom software contracts over $10 million. Just don't allow them. If a vendor is truly excellent, give them the full $10 million and see how it goes. If that goes well, give them another $10 million. But don't write one giant check up front.
3. Require user research for any project over $1 million. That means a trained professional should observe the people who will use the system, confirm whether the need is real, and validate that the proposed solution will address it. If it was up to me, I'd give ADS the funding and the mandate to hire those user researchers.
4. Every project needs a single product owner—one person, a full-time employee, empowered to make decisions and be accountable for the results. Right now, no one is on the hook.

I have no doubt that ADS could lead that, if empowered to do so and given the resources.

And here are three actions the legislature itself can take:

1. Provide oversight by attending some of those twice-monthly demos, especially for high-value, high-risk projects. Insist on seeing actual software as it's being developed. If there's nothing to see every two weeks, the project is probably going to fail. Make that a strict requirement.
2. Expand the mission and capacity of ADS to include this sort of work. Generally this looks like a small team made up of coders, designers, user researchers, and a contracting officer. Their job is to help agencies succeed at this work: conducting user research to validate that needs are real and proposed solutions are viable, writing and reviewing RFPs, helping to select vendors, and verifying that vendors are delivering. This is not expensive, and it pays for itself many times over by preventing failures. A handful of states already provide these teams, and in the past couple of months there's been an explosion of interest in establishing them. Pennsylvania, Colorado, and New Jersey are good models to follow.

3. Budget for software as an operating expense, not a capital expense. Custom software development is not a one-time activity. It's an ongoing process—a team of people doing work for a predictable cost, over time. That's not a capital project.

One final thought: If the technology fails, the legislation fails. So don't let the technology fail.

You can't expect success if you treat technology as a separate thing that can be bolted on at the end. That doesn't work. Technology undergirds nearly all government initiatives now. If the tech fails, the initiative fails.

Legislation is meaningless if it depends on technology that agencies are powerless to implement reliably.

As long as legislators see technology as an uncontrollable externality—a pit into which money must be shoveled until the vendor says "that'll do"—then important programs will fail. New initiatives won't get off the ground. Good bills won't pass because of the high cost of implementation.

Technology can be a blocker to progress—or it can be the thing that makes progress possible.

A new approach to budgeting and oversight is necessary to get this right. Everything I've told you this morning can work in Vermont. There's nothing I've told you that requires a large population or a huge budget. You have everything you need in Vermont to move to an IT funding model that can reduce risk, allow oversight, and ensure that state technology systems can deliver on public needs.