⚕ ⚛ David Farley – This Government Software Project WASTED $500,000,000... Here's Why [10 Jan 2024]

2088z6) ⚕ ⚛ David Farley – This Government Software Project WASTED $500,000,000... Here's Why [10 Jan 2024]

By **David Farley**

Saturday – 12:11 GMT – January 11, 2024

Bullet points from David Farley's video

341 Words:

START: *This Government Software Project WASTED $500,000,000... Here's Why*
https://www.youtube.com/watch?v=w-_nCA3RTYs

- **The Standish Group estimates that 87% of government software projects over £6 million fail.**
  - The Fire and Rescue Services project budget was £120,000,000, but it cost £469,000,000, and no working software was ever delivered.
  - This project was imposed on its prospective users, who didn't think it was a very good idea from the start.
  - It focused on process adherence rather than achieving the goal.
  - It spent an inordinate amount of time getting people to status-report against meaningless metrics.
  - Despite the scale of failure and waste, no one in the department has been held accountable.
- **Waterfall-style development:** The assumption that we can develop a perfect plan deeply misunderstands what software development is all about.
  - Surely, when beating your head against a wall, there should come a point when the blood and pain tell you to stop and try something different.
  - It's not about the act of coding. It's much more about an ongoing process of learning and discovery and is really about continual learning.
- **If we can make a project smaller and simpler, that's what real success looks like.**
  - Fundamentally, the route to successfully building any complex system is to begin with a simple system and make it work first. Organise to learn from that, start with a system that's easy to change, and then change it incrementally, step by step. Learn what works until you get to something genuinely useful, and do more of the stuff that works instead of what doesn't.
  - This is the only proven route to success. It is, after all, how all science and engineering works. Find out what really works and what doesn't. Even if we disagree when we start, we can discover which viewpoint is correct and adapt our understanding and systems to target the solutions that work. This doesn't rely on armies of people, but it does take a different perspective on the nature of software development and what it takes to make it work.

Saturday – 19:56 GMT – January 11, 2024

Extended Bullet Points from David Farley's Video

START: *This Government Software Project WASTED $500,000,000... Here's Why*
https://www.youtube.com/watch?v=w-_nCA3RTYs

- **The Standish Group estimates an 87% failure rate** for government software projects with budgets over $6 million.
- **A profoundly broken model** of what software development is really all about exists in many organisations, particularly in government.
- **The Fire and Rescue Services project:** Budgeted at £120,000,000 but ended up costing £469,000,000, with no working software ever delivered.
- **Focus on process adherence rather than achieving the goal:** Many projects measure success by adherence to process rather than meaningful outcomes.
- **Spent an inordinate amount of time** on status reporting against meaningless metrics.
- **"It's not about the act of coding."** Software development is much more about **ongoing learning and discovery**—a process of continual adaptation.
- **Big complex software projects are inherently uncertain:** This may feel disquieting, but it is deeply true. Effective projects embrace this uncertainty.
- **This project was imposed on prospective users** who didn't agree with the premise, the solution, or the approach.
- **The assumption that a perfect plan is possible** deeply misunderstands the nature of software development.
- **Waterfall-style development:** A traditional approach that almost never works, yet it is still the default for most large projects, especially in government.
- **"Surely, when beating your head against a wall..."** There should come a point when the blood and pain tell you to stop and try something different. Farley humorously highlights the correlation: *"We always do things like that, and 87% of projects fail."*
- **No accountability despite failure:** Despite the scale of waste, no one was held accountable. **Contractors and individuals responsible often face no consequences** and may even receive promotions or further lucrative contracts.
- **Overcomplicated staffing:** One government project had nearly **1,000 technologists from over 30 competing, sometimes antagonistic contractor organisations.** This is not a recipe for success.
- **Effort is often misdirected:** In such environments, more time is spent **avoiding accountability** than building good software.
- **Throwing more people at the problem doesn't work:** The assumption that adding more staff will speed up a project and lower costs is fundamentally flawed and disproven since the 1970s.
- **Middle management incentives drive poor decisions:** Success for some managers is measured by the size of their budgets and teams, leading them to **inflate projects unnecessarily** to enhance their reputations.
- **Real success lies in making projects smaller and simpler.**
- **The proven route to building complex systems:**
    - **Start with a simple system** and make it work first.
    - **Organise to learn from that system,** keeping it easy to change and improving it step by step.
    - **Keep the system working continuously,** learning as you go.
    - **Do more of what works** and stop doing what doesn't.
    - **Adapt to new information** as it challenges assumptions, refining both understanding and systems over time.

- **This is the only proven route to success.** It mirrors how all science and engineering works, software or not.
- **User-focused, incremental problem-solving:** By fixing real user problems incrementally, users can join the learning journey, even if they initially disagree with the solution.
- **Success doesn't require armies of people:** It requires a fundamentally different perspective on what software development is and how to make it work.

A comment from youtube:

@tullochgorum6323
Let's name names - the lead contractor for the failed FireControl project was Siemens. Why governments continually use large, faceless conglomerates for these contracts is baffling. You'd have thought that they'd have learned from the £10 billion failed NPfIT project and the catastrophic Fujitsu Horizon project, but they make the same mistakes again and again. The alternative would be to assemble small teams of elite developers to drive a highly focused, user-centric adaptive process. But this is never done.

GPT-4o Transcription Perfected

David Farley
This Government Software Project WASTED $500,000,000... Here's Why

https://www.youtube.com/watch?v=w-_nCA3RTYs

Government IT projects don't have a very good reputation; they're often headline fodder and seen as big, expensive problems. So why are government projects so complicated, so often prone to failure, and what can we learn from them? This is true even if we don't work on government projects. What could we do to achieve better results?

Hi, I'm Dave Farley of Continuous Delivery. Welcome to my channel. If you haven't been here before, please do hit subscribe, and if you enjoy the content today, hit like as well.

I guess that the first question to ask is: do government software projects really fail more often than private sector projects? The answer appears to be a resounding yes—they do. One report from the Standish Group estimates that only 13% of government software projects with budgets over $6 million succeed, leaving an 87% failure rate. That's not doing well, really.

Most reports seem to put these failures down to some common reasons. Here's one such list. But if you look at the scale of government projects, they're often not especially large, at least compared to other modern global systems. Although it's hard to compare different systems in this respect, they're not obviously unusual in terms of their complexity either. For example, is the coordination of the work of the fire and rescue services in the UK really inherently more complex than planning the logistics of, say, a shipping company, running a bank, providing public cloud services like AWS, or search like Google? Or, for that matter, hundreds of other complex problems that the private sector has addressed seemingly with more success?

The other question is probably—unless you're already working on government software projects—why should you care? I think there are a couple of important reasons to answer that question.

The first and perhaps most obvious is that we, as taxpayers, pay for these projects and are usually, at least at some level, the people that these projects are meant to serve in some manner, and so we are certainly stakeholders.

But of wider significance, I think that the problems of government software projects are not really special to government projects. Rather, they're pretty widespread in non-government projects too. The stuff that I talk about in this episode is common in many big organisations that create software, government or not. As well as examples of bad practice in the private sector, there are also obviously examples of good work inside government too.

Government software projects do, though, have some additional characteristics that seem to systematically make the impact of these problems worse. But the causes of failure and patterns of poor results are commonly seen in projects everywhere. So maybe there are some lessons here in these more extreme cases and the greater susceptibility to failure of government projects that we can all learn from.

One of these generic causes is that most people—and certainly most organisations—have a profoundly broken model of what software development is really all about, and so organise it very poorly indeed. The really frustrating thing is that, while this is seen as a failure of IT projects, it actually has very little to do with information technology or software development, at least not directly.

Let's look at an example of these government projects.

The UK government started a project in 2004 to help better coordinate the work of fire and rescue services across the country. This project inevitably failed six years later. The original budget was estimated at £120,000,000, but by 2010, when the project was shut down, they'd spent at least £469,000,000 and had nothing left except nine large buildings—eight of which had never been used for anything and were costing millions per year to maintain—and no working software was ever delivered.

This is a pretty extreme example, but not a terribly unusual one. The parliamentary investigation found the following list of causes for the failure, which I'm sure are true, but, to me, they still seem to be talking more about the symptoms and missing the real causes and how to address them.

Main Findings

1. **The department failed to secure the cooperation and support of local fire and rescue services.**

   This is a pretty classic problem in big software projects, but probably in big projects of all kinds, to be honest. It's all too easy to lose focus on the goals and instead get lost in the process of delivery. Instead of worrying about, focusing on, and understanding the problem, we jump to conclusions about a solution—usually to a problem that we've invented—and then measure our progress against artificial indicators, typically focused on process adherence rather than on measures that actually show whether we're getting closer to or further away from our goal.

   Following a process is irrelevant if it doesn't achieve the goal. Statements like, "We delivered X features this week," or "We had a stand-up meeting every day," or "We're on track with our Gantt chart," are not measures of success. This doesn't mean that process is unimportant, but it does mean that process is a means to an end—not the end itself. Process adherence is never a good measure of success, nor is it necessarily a measure of being on the best route toward

success. You can stick to the process perfectly and still not deliver anything of value, particularly if you've chosen the wrong process to begin with.

2. **The department failed to apply effective checks and balances from the start.**

   This is a fair criticism, but it does rather imply that we know what the right checks and balances are from the start. The truth is, we never do. This assumption represents a profound misunderstanding of the nature of software development. It's not about the act of coding. It's much more about active learning—understanding enough to decompose the problem in sufficient detail to be able to code.

   This means working in ways that allow us to evolve and correct our understanding as we go. It doesn't rely on perfect predictions of the future. Software development is an ongoing process of learning and discovery. Organising to facilitate that learning is much more important for achieving success than trying to correctly identify checks and balances at the outset.

   This may feel disquieting—that big, complex software projects are a process of exploration and uncertainty—but it is deeply true. Organising on any other basis is, at least to my mind, an exercise in delusion, which may explain the 87% failure rate.

3. **The department's management and oversight of the project were weak.**

   Personally, I doubt that this was the real problem. This criticism implies that everything would have been fine if we'd only micromanaged it in more detail. Sure, management and oversight are important, and in this case, they may have been unusually poor. However, even if they were good, management in large organisations is often interpreted to mean spending an inordinate amount of time getting people to status report against meaningless metrics. That's not going to help.

   The real name of the game is figuring out what the actual problem is and solving that problem.

4. **The department did not approach the project as one of business transformation but instead treated each element in isolation.**

   This is clearly a real problem in this case. It was a typical ivory tower project imposed on its prospective users without considering the reality that the users didn't agree with the premise, the solution, or the approach—and, generally, didn't think the project was a very good idea right from the start. Again, this reflects a lack of focus on the outcomes.

5. **The department failed to manage the delivery of the IT system by the contractor.**
6. **The department failed to follow the most basic fundamentals of good contract management.**
7. **The department's wider corporate governance arrangements failed to provide an effective check on the project.**

The next three findings are really shouting at us about the limitations inherent in the assumed approach to the project—an approach so common in big projects in general and big government projects in particular. This is the assumption that we can come up with a perfect plan, and, based on that perfect plan, have enough understanding to draft a perfect contract, so we can then outsource the development to a bunch of other people.

That assumption depends on the idea that this is a problem amenable to some form of software production line where we can perfectly specify what everyone needs to do. This simply and deeply misunderstands what software development is all about.

Of course, big complex projects need effective management, but my suspicion is that what's in the heads of most big organisation people when they read things like "effective management" or "good contract management" is **waterfall-style development**—the traditional approach that almost never works yet seems to be adopted by nearly all large projects by default, especially big government projects.

Surely, when beating your head against a wall, there should come a point when the blood and pain tell you to stop and try something different. Maybe something like opening the door that's next to the wall—the one everyone else is using—rather than trying to knock the wall down with your head.

It seems to me that one of the reasons large government projects are so prone to failure is that the politics and social structures in and around government tend to push people into making dumb decisions. Because we've always done things like that. Surely there's a correlation between the fact that we always do things like that and the fact that 87% of projects fail. Wall, meet head.

8. **Despite the scale of failure and waste, no one in the department has been held accountable.**

Perhaps this last item in the findings of the select committee is the most telling one. Where is the feedback loop to help correct things if the contracting companies and individuals responsible for failures on this scale never suffer any consequences for their lamentably poor choices?

Not only has no one been held accountable for this project's failure to this day, but many of the principals involved received promotions, and no doubt the contractors went on to win many further lucrative government projects.

I once saw a large government project reasonably close up. It was staffed with nearly 1,000 technologists from over 30 different competing, sometimes antagonistic contractor organisations. This is not a recipe for success. More time and effort seemed to go into not being held accountable for anything than into actually building good software.

The social and cultural pressures inside government are problematic. At one level, people are afraid of being found to be wasting taxpayer money, so projects are micromanaged to ensure costs are kept low, often by cutting corners everywhere. But confusingly, projects like this are also treated as some kind of exercise in massive-scale civil engineering, with armies of people being thrown at problems that would be better solved by dramatically smaller, more outcome-focused teams.

I suppose the generous interpretation of this decision to throw people at the problem is that it's based on the ridiculous assumption that this will allow the project to go faster and thus be cheaper overall. This, though, is despite the understanding—known in professional software development circles since at least the 1970s—that this is not the case.

So why are ridiculous decisions like this made so often? Well, a less generous interpretation—which is certainly a common factor in some big organisations—is that, for some tiers of middle management, their success is measured based on the size of the projects they lead and the size of the budgets they control. So, guess what happens? People leading projects like these throw people at

the project to enhance their own reputations. If they don't do this—even if they succeed in smaller, simpler projects—their success is wrongly attributed to the problem being small and simple.

The reality, though, is quite different. If we can make a project smaller and simpler, then that's what real success looks like.

Based on the data, the reality of big government software projects is that if you gave my team the job, and we charged half the original estimated price and then went on holiday for the rest of our lives, we would have saved the government £409,000,000 and achieved exactly the same result: nothing.

The list that I mentioned earlier rings very true to my ears. Fundamentally, the route to successfully building any complex system is to begin with a simple system and make it work first. Organise to learn from that, then maintain the starting simple system so that it's easy to change, and then change it incrementally, step by step, keeping it working all the time, learning all the time.

Learn what works for your users and what doesn't until you get to something genuinely useful. Learn what design works and what doesn't, and do more of the stuff that works instead of the stuff that doesn't. Learn how best to organise and apply team technology solutions, and change what you need to as you learn new things that challenge your assumptions—wherever they may be.

This is the only proven route to success. It is, after all, how all science and engineering works, software or not. Only this way can we figure out what really works and what doesn't. And if we take the opportunity to fix the real problems that users face and do that incrementally, we can also bring them along on our learning journey with us.

Even if we disagree at the start about what the problem is and how to solve it, we can discover which viewpoint is correct and adapt our understanding and our systems to target the solutions that really work. This doesn't rely on armies of people, but it does take a very different perspective on the very nature of software development and what it takes to make it work.

~

If any of this sounds surprising, welcome to the Continuous Delivery channel. Do remember to hit subscribe, and maybe check out my introduction to Continuous Delivery—there's a link to that in the description below.

Thank you very much for watching. If you enjoy our content on the Continuous Delivery channel, do consider supporting us by joining our Patreon community. There's a link to that in the description below.

As usual, thank you to all of our existing patrons for your ongoing support—it's very, very much appreciated. Thank you.

Link for reference:
https://www.objectstyle.com/blog/7-reasons-behind-costliest-government-project-failures