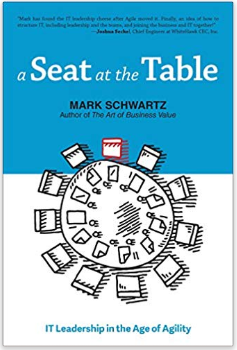
A Seat at the Table – IT Leadership in the Age of Agility – Part 2

By Mark Schwartz

“To the talented and hard-working government employees, so resilient in the face of impediments, criticism, and abuse. I have so much fun working alongside you.”  
 – **Mark Schwartz**

# Last Time in Part One

**Planning:** The idea that we should make a plan and then stick to it is a terrible idea in an environment of uncertainty and change. It has dominated the IT world because it appears to offer predictability, control, and efficiency, the key values of the contractor-control model. But it doesn’t.

**Requirements:** Requirements are a way of controlling the development team by constraining their creativity. Instead of requirements, we want to charge the team—the joint business/IT team, that is—with delivering business outcomes, and let them find the capabilities that deliver these outcomes best.

**Transformation:** Transformation and modernization projects are exactly what IT leaders must avoid; continuously transforming and modernizing the company’s IT systems makes Fowler’s strangler pattern into an IT strategy rather than just a coding tactic.

Posted on the Agile4Defense GitHub page at: <https://git.io/JeaOu>

# Enterprise Architecture

Enterprise Architecture, the domain of the IT bureaucrats, is the place we must look for the solution to our Agile challenges. We shall journey to the land of the template zombies to retrieve our golden asset, careful to carry mirrors to avoid petrification. Good luck, Agile fellows.

**In the past:** We viewed EA as primarily concerned with standardization, consistency, planning, and cost reduction. It documented as-is and to-be architectures, demonstrated alignment of systems with business needs, and did the “rigorous” up-front analysis and centralized planning that could then be used to set boundaries for developers when they began a project. In other words, a vehicle for control. But standardization also imposes costs by:

* limiting agility and adding bureaucratic waste: exceptions must be put through an approval process
* standardization limits the space of possible solutions to a problem.
* if we mandate that projects reuse code whenever possible, each project may have to spend time searching archives of available code to find something that is a near fit, and then deal with the question of whether it is a near enough fit.

I don’t mean that standards are bad. Let’s just agree that they might be overrated.

**A Better Way – Treat IT as an Enterprise Asset (EA):** When we add all of our current IT capabilities together, we arrive at an asset that enables the enterprise to earn future revenues and reduce future costs—that is, an asset in the classic economic sense. This asset I will refer to as the EA, which could just as well stand for Economic Asset.

* The EA has intangible, latent capabilities—potential that is, for the moment, hidden.
* The EA asset evolves over time through incremental investments.
* Managing the EA asset is an art, just as all strategic management is an art. Just as the CMO must sense market opportunities, weigh tactics for communicating with that market and encouraging it to purchase, and manage creative talent to present the message compellingly, the CIO must similarly steward the EA.

What does such an EA, brimming with latent value, look like?

* It has little technical debt.
* It is built according to good, extensible design patterns and uses well-accepted standards.
* It is loosely coupled—pieces of it can be easily exchanged for newer pieces without requiring changes to the whole ball of EA.
* It has a robust, automated regression test suite, so that new development does not cause expensive break-fix activity.
* It has good monitoring tools in place.
* It is coded in a way that resists hard-to-find defects like concurrency errors.

# Build Versus Buy

Everyone knows that in every case under the sun, in any example one can imagine, when rational human beings are making decisions, if an IT product can be acquired “off the shelf,” it is better to do so than to build it. This obvious fact is neat, plausible, and in most cases, wrong.

**The economics of software development have changed:** Changed in a way that now favors “building” over “buying.” There are now ways of custom-developing systems that preserve many of the advantages of buying off the shelf.

* The risk of developing a system incrementally and altering it based on user feedback is often lower than that of buying a finished product that is hard to change.
* The advantages of the agility that can be gained through a flexible, changeable, custom system—a smooth rather than a lumpy EA, as we put it in the last chapter—are becoming more compelling, and the disadvantages of proprietary products, always evident, are becoming harder to accept.

**We make this mistake a lot:** We organize our business around a product rather than creating IT capabilities to fit our business.

* We acquire a product that does not fit our business and customize it until it does.
* We offer our business users a user interface that is clunky because it was designed for users in the abstract across many possible companies and usage situations, rather than being designed for the particular needs of our employees.
* Off-the-shelf systems are more expensive than we expect, and take longer to roll out. We pay for features that we don’t actually use.
* It resists change—we can’t even change it ourselves, but have to get the vendor to change it.
* Our IT Skills Asset also becomes less flexible when we acquire an off-the-shelf product, since we will have to employ people with skills in that product.

**The cost of custom development is falling:** More and more logic is abstracted away by frameworks and design patterns. Incremental delivery and staged investments reduce cost and risk.

* Custom code is almost not custom these days. A developer incorporates open source frameworks, uses standardized design patterns, and orchestrates services that are already available.
* There are “cookbooks” available with templates for deploying systems, code snippets that handle common tasks, and well-known and well-studied algorithms for solving typical problems.
* Development takes place on automated pipelines that help the developer move frictionlessly and quickly from requirements to deployment.
* The developer uses a powerful integrated development environment (IDE) that centralizes the tools that help him or her do the job.
* The code is built automatically using Continuous Integration, and it is tested automatically through scripted tests.
* It is deployed automatically with automated deployment tools onto infrastructure that can be automatically provisioned.

**The choice we have to make:**

* Option 1: These techniques, if we use them correctly, enable a fast try-and-learn cycle in which developers can produce something, get feedback, and then adjust what they have produced. As a result, the code can be developed in a user-centric way and match the enterprise’s needs precisely. Changes are easy to make with automated regression tests that allow experimentation without fear. Risk is low, because the team is constantly adjusting.
* Option 2: Compare that to the risk of buying a vendor’s product, where the investment is one large lump sum—and a commitment to future maintenance payments. Then, of course, there is the risk of the vendor going out of business or discontinuing the product. It seems strange that back in the Stone Age (yesterday) we believed that it was riskier to develop custom code.
* So, today’s choice is no longer really between build and buy. It is between quickly assembling best-practice frameworks with continuous user feedback and then continuing to adapt the system over time as the business changes versus buying an undefined stream of future services from a vendor who doesn’t know your business and doesn’t have financial incentives to support you. Text “1” to @obvious if you like the first option, or “2” to @/dev/null if you prefer the second.

# Governance and Oversight

Governance has traditionally been viewed as a filter; a way of allocating scarce IT resources among many competing projects. In an environment where IT has a limited capacity and the company’s needs for IT are insatiable, governance provides a way to say no to proposed projects, absolving IT leaders from having to take personal responsibility for such decisions.

**Traditional governance approach:** Hunter and Westerman’s book *Real Business of IT* shares the traditional view of governance.

* The basics of the [governance] process involve project sponsors

1. developing a formal proposal that incorporates estimated benefits, risks, and resource requirements and
2. submitting the proposal to decision makers who select preferred investments from the proposals ... the first question to be answered in assessing a proposed initiative is this: exactly how, and how much, will the investment affect and improve business performance?

**Mark’s reality check:** I imagine a sort of Star Chamber—a dimly lit group of serious, hooded faces (“decision makers”) seated around a table, passing judgment on each “formal proposal” presented to them. That last piece of Hunter and Westerman’s quote gets me every time. Exactly how and how much will the investment improve the business’s performance? Are they serious?

* Do we know exactly how many consumers would buy our new product?
* Exactly what price they would pay?
* Exactly how much it would cost us to design and produce it?
* Will our competitors launch a competing product?
* Will our market be wiped out in a dramatic fireball from space, or will the Four Horsemen ride in just as we start our TV commercial campaign?

I will turn off my sarcasm engine for a moment. Look—it turns out that the future involves lots of uncertainty. No one knows exactly what the benefit of an investment will be—no one even knows approximately what it will be—in fact, I doubt that anyone could agree exactly on what they even mean by “benefit” and how they would measure it. Yeesh. What a basis for making such important decisions!

These Waterfall governance and oversight practices are not only inconsistent with Agile principles, but also fail to take advantage of the power of the Agile way of working. By proceeding with agility:

* We can reduce the risk of our governance decisions—once we admit that there is substantial risk in making decisions in an uncertain world, the subject of the next chapter—by quickly deploying functionality, learning, and adjusting plans.
* We can conduct experiments that test the assumptions in our business cases.
* We can reduce the overhead of our governance processes and make smaller, incremental decisions with correspondingly lower risk.
* Lastly and critically, we can encourage the teams that create value for the company rather than standing in judgment of them.

**Characteristics of an Agile governance and oversight model:** Before we dive into an Agile governance and oversight model, let’s think about what characteristics such a model should have in order to both take advantage of the Agile mindset and remain consistent with it.

* the investment decision and the oversight process would form a seamless continuum wherein the execution of the initiative would yield valuable information by which the governance decision could be adjusted. We would manage risk by only committing resources to the smallest piece of work that would give us such useful learning.
* we would gauge progress by seeing operational results. Agile and Lean approaches allow teams to quickly put product in the hands of users. This not only delivers value immediately, but also lets us validate the features that have been deployed.
* we would carefully set boundaries for planning. A detailed plan is a less effective basis for governance than validated learning based on actual delivery. The more advance planning we do, the longer it takes to get a product to market; the longer it takes to get a product to market, the more risk we assume.
* our governance decisions would not be based on a set of required requirements, but instead on high-level objectives. Instead of telling the project team how to do its job, we might begin the project with a hypothesis of what will best accomplish the objectives, but then allow requirements to change. Based on what we learned, we would re-validate these hypotheses.

**Oversight in the Federal Government:** In speaking to the government officials responsible for overseeing our IT programs—our very own Star Chamber—I frequently heard them say things like, “I want you to tell me what you are going to do and when you are going to finish, then I want you to tell me your status and what is left to do. I want to know exactly what I will get for my money and when.”

* My team would respond by producing its Agile interpretation of what the overseers were asking for. That might include lists of user stories, burndowns, definitions of done, product roadmaps, and, well, in the government, about 100 different piles of paper.
* Now, I believe that this was a dead end: it was a Waterfall solution to the problem.

**A Better Way - Develop an Agile oversight process incrementally:** The overseers were right to demand what they did—they were effectively the customers of our Agile governance process, and we had to find a way to satisfy them as customers. When it came to making the governance decision, what the Star Chamber really cared about was whether they believed the money would be well spent, and spent on an initiative that was important to the agency. What we really had to show the Star Chamber was

* that the desired outcome represented a critical goal for the agency,
* that the outcome justified the spending that was proposed,
* that the team was capable, and
* that the Star Chamber would frequently be consulted and would be able to influence the team’s actions, including changing or eliminating the budget.

Now, the first governance and oversight step.

* We presented to Our Very Own Star Chamber the proposed outcomes and budget, some justifications of why we believed it was plausible, and just enough description of the execution team and its process to convince the Star Chamber that the team was capable.
* Then, we did something unprecedented in formal government oversight—we had a discussion. The Star Chamber had the chance to give feedback, steer the program in different directions, and ask that the budget be changed. It was a full participant, not just a passive judge of proposals.
* We agreed to give Our Very Own Star Chamber a monthly one-page summary of the program’s status (more on that later) and appear quarterly for a more detailed discussion.
* At any time, they could call us in if they saw something alarming in the monthly update. With that, the program was approved to begin.

Now the focus shifted to oversight at my level, and, as with many of our Agile programs, we set up a regular cadence for reviews.

* Every two weeks, we would have a quick meeting between the program team, me, and the business unit leader who was my peer. The main purpose of these meetings was to laugh a lot, make fun of the Star Chamber’s serious facial expressions, and enjoy the successes we were having—in other words, an opportunity for, um, “vigorous discussion.”
* Every quarter, I would do something I called a release cycle review, and then we would go back to the Star Chamber for an update discussion.

**How to gauge success:** To gauge the success of the project to date, I need to know only two things.

* The first is what the initiative has delivered so far. Delivery refers only to finished, deployed capabilities that are being used by the business, and I am especially interested in any measures of the delivered value of those capabilities. To show me what the initiative has delivered so far, the team prepares a Value Delivery Register—a high-level summary of the value that has been delivered.
* The second thing I want to know is how much we have spent so far. Spending in this case is measured in dollars, or in time, or in whatever other metric is most appropriate (what we are typically looking for is opportunity cost). We use a simple, one-page document we call the Financial Summary for this purpose.

**How to plan:** To gauge the plans for the future, I again need to know two things.

* I need to know, as of now, to the best of the team’s knowledge, what value it thinks it will deliver in the future—its roadmap—more detailed for the near future and more coarse-grained and tentative for the long term.
* Then I need to know what its high-level budget is for that future work.
* By comparing these two things—roadmap and budget—I can decide whether the business case for the investment is still attractive. Often, the plan for the future is to continue exploring the ideas originally brainstormed in the impact map. That makes it easy.

# Part Three?

**Risk:** The presence of uncertainty is the simple reason why Agile approaches work better than plan-driven approaches—it is also the reason why a good IT leader will often have to make “wrong” decisions. An IT leader adds business value by adopting an intelligent attitude toward risk.

**Quality:** It is difficult for IT to gain a seat at the table when IT is always failing, but on the other hand, an IT leader who is reacting to statistical noise—failures that he or she has already chosen to accept—is destroying business value. An IT leader must have the necessary technical skills, make impeccable decisions under uncertainty, and then have the courage to face the consequences.

**Shadow IT:** Agile ways of working support a community approach to IT, where IT leaders achieve their objectives by mobilizing the skills and passions of a broad community and encourage the members of that community to work together across organizational silos in a way that values skills and contributions.

**Exhortation and Table Manners:** There’s nothing left to do but end on a message of hope. And that’s what this book is meant to be, really.

# About the Author

Mark Schwartz is an Enterprise Strategist at Amazon Web Services and the author of The Art of Business Value and A Seat at the Table: IT Leadership in the Age of Agility. Before joining AWS he was the CIO of US Citizenship and Immigration Service (part of the Department of Homeland Security), CIO of Intrax, and CEO of Auctiva. He has an MBA from Wharton, a BS in Computer Science from Yale, and an MA in Philosophy from Yale.

Mark and Mike first met at the 2019 AWS Public Sector Summit in Washington, DC. They’ve been best friends ever since.

# Fork Me on GitHub

AgileForDefense on GitHub: <https://github.com/Agile4Defense/AgileForDefense>

This paper on GitHub: