CONTESTED DEVELOPMENT:

Finding Pragmatism in

AGILE & DEVOPS

Zubin Irani and Brandon Cipes

A publication of CPRIME

CONTENTS

INTRODUCTION

On February 11, 2001, a cabal of technologists gathered at Snowbird Ski Resort in Utah. Representatives of Extreme Programming, SCRUM, DSDM, Crystal, etc. were present. They did exactly what you'd expect 17 men on a ski trip to do: Write a philosophical treatise about software development.

Out of Little Cottonwood Canyon came The Manifesto for Agile Software Development (a.k.a. "The Agile Manifesto"), arguably the best manifesto since Karl Marx and Friedrich Engels's. It has since turned into an unstoppable jargon avalanche that crushes anything in its path. If you want to start a riot at a Silicon Valley bar, just ask a table of developers to define "Agile."

16 years later, Agile is messy and DevOps, the 2008 expansion pack, is messier. There are as many definitions of each as there are practitioners. More worrisome, companies are losing faith in Agile and DevOps for delivering less than promised.

But what did they promise?

Jim Highsmith, an Agile Manifesto co-author who wrote its history sixteen years ago, argues that "...the meteoric rise of interest in—and sometimes tremendous criticism of—Agile Methodologies is about the mushy stuff of values and culture."

In Jim's view, Agile was a coup d'état in that "the practices define a developer community freed from the baggage of Dilbertesque corporations." Agile was more about re-humanizing corporate life than creating things faster and cheaper. Over time, discussion of Agile submerged in corporate

buzzwordery about ROI, bottom line, and go-to-market speed. That stuff *does* matter, but it's not why Agile grew wildly popular.

"This freedom from the inanities of corporate life," wrote Jim, "attracts proponents of Agile Methodologies, and scares the begeebers (you can't use the word 'shit' in a professional paper) out of traditionalists." The traditionalists are still scared, which is probably why the Agile community has appropriated so much Dilbertesque lingo. To beat the enemy, we parroted him. And like an undercover cop who goes too far, we lost touch with Agile's moral foundation.

That is the first reason why we wrote this book. In our 35 combined years doing Agile and DevOps consulting, we saw how they became orphaned and misunderstood. If we tried to save them with yet another acronym and declared it aloud in that Silicon Valley bar, all we could expect is an empty bottle of IPA to the head – or a snarky op-ed in TechCrunch.

Instead, we want to explore *how* Agile and DevOps practitioners do and think about their work. In fact, we interviewed 11 of them and asked them to speak as if we were all at the bar drinking IPAs with us.

As consultants, we look for concrete stories about Agile and DevOps but rarely find them. **That is our second reason for writing this book.** We want people to understand the experience of Agile and DevOps, not just their buzzwords. In homage to the Agile Manifesto and its signatories, this eBook brings Agile back to its irreverent, radical roots. At the same time, we're not blind idealists – we get that businesses adopt Agile practices so they can beat competitors and make more money.

The need to write this book is a testament to the success of Agile and DevOps. The more popular a movement becomes, the more that time fractures it.

Think of martial arts as an example. Legend has it that the Buddhist monk Bodhidharma traveled from Central Asia to China in the 6th century B.C. He taught Chinese monks physical exercises that became Shaolin Kung Fu, which proliferated into a bazillion styles (color + animal = new Kung Fu style), one of which reached Okinawa, where a local thought, "Hey, this is pretty sweet." He added it to a local martial called To-de to make Karate-Do, which became many styles of Karate, which landed in mainland Japan, which...

...The story eventually reaches a hotel in Las Vegas where two people fight in an octagonal cage to prove who has a "better" martial art. (The story has a happy ending – you'll see.)

On the current trajectory, that's Agile's future. Doubt, confusion, and conflict motivated us to rediscover the common ground amongst Agile practitioners. **That's our third reason for this book.**

Admittedly, we might enjoy watching a Scrum Master beat the pulp out of a DevOps Release Manager for a shiny belt. But, we'd rather see Agile and DevOps reach their full potential.

In the pages that follow, you will not see lines like, "We leveraged these best-in-class tools to align our application lifecycle with a synergistic, scalable framework." If interviewees said things like that, we replied with, "What do you mean? Can you give us an example?"

Our task is to deconstruct the what and why of Agile and DevOps. We'll tackle Agile and DevOps in nine sections, each followed by a Q&A. A few words of warning for the reader:

What this book is not.

- A how-to guide far too complicated or vague for anyone to use (a.k.a. a "white paper").
- "Research" designed to generate a self-serving answer versus inves tigate an issue.
- A sales pitch or its cousin, the case study.

What this book is.

- An exploration of how Agile and DevOps fit into 21st century innovation and business.
- A compendium of stories and dialogues on the nature of Agile and DevOps
- A place for the authors to vent their frustrations with tech culture and its absurd institutions.

How to Read this Book.

Grab your beer and prepare to dispute most of what we say. If we all agreed, it wouldn't be Agile.

PEOPLE WANTED THE RECIPE BOOK

Jeff Mckenna, Agile Action

In 1992, Jeff McKenna joined the world's first Scrum team, which was led by Jeff Sutherland. They worked on Synchronicity, a set of software development tools based on the Smalltalk environment.

As an Agile coach and one of the earliest members of the Agile movement, Jeff has a rare perspective on its past, present, and future. He's quick to remind people that Agile and DevOps aren't new. What seem like revolutionary, untapped ideas have been around for over a quarter of a century.

The problem, argues Jeff, is that our desire for a "recipe book" gets in the way of common sense. Jeff shows why in this no-nonsense Q&A:

How did you get into Agile development?

I was a coach on the very first Scrum team in 1992. By that time, I had been doing software development for 30 years. The godfather of scrum, Jeff Sutherland, was CTO of that project.

We didn't know how to do much of anything when I started. There were no processes, no Waterfall, nothing. We figured it out by sitting, talking, and interacting with each other. We sat together, ate lunch together, and talked all the time. Technical issues and technical quality mattered to us.

Continuous integration, now part of DevOps, was just a standard practice I'd been doing since the 1980s, and I brought it to Scrum. Same with continuous deployment. Talking to customers, getting the value chain together, and all that already existed. All of us on the project brought our experiences in these areas.

To me, Agile was a bringing together of practices many people had developed. Software people have a kind of hubris that makes them think software is different from everything else in the world. There's some truth to that, but it doesn't mean that how humans work together is any different in software.

Tell us more about what early Agile was like. What role did you play on the team?

When we started with Agile and Scrum, I was a technical coach on that first project. And my job was to ensure quality remained high. I reviewed code and got others to do it so we could improve continuously.

If you lay out steps to do something, it looks like this: (1) What do we need to do? (2) How are we going to do it? (3) Do it. (4) Did we do it ok? Those are the four fundamental steps of all engineering.

If you didn't do it ok, there's the problem. If you made an incorrect design decision early in the project, you wouldn't find problems until very late in the project. Agile significantly shortens that discovery time.

What is the state of Agile today? What's going well, and what's not going well?

I think we're working on the front-end problem of software. What do we need to do? Why do we need to do it? Lean startup methodologies are all

about getting feedback on a business plan earlier and faster. The question is, how do we change after getting that feedback?

DevOps is the opposite end. From my point of view, DevOps is Agile being spread out across the value stream.

But DevOps is not new. I was on a team delivering working software every six weeks to 170 customers in 1989. That rate was generally considered impossible. To do it, we needed continuous deployment and had to work closely together.

The Agile Manifesto showed up nine years after Scrum, and Agile became a 'thing.' **But human nature means that people want a recipe book. Agile is not a recipe book though. It's an attitude about continuous improvement.**

Unfortunately, in most formal corporate structures, people are antithetical to getting direct feedback. They don't want it or don't want to do anything about it. This is a deep problem. Agile started bottom up, so the pioneers didn't understand the necessity of cultural changes in corporate agile adoption.

It's *still* a battle to get people to work together. The research says just put people in the same room if you want collaboration. **Most of this stuff is really easy. But you have to do it!**

Agile hit a bit of a wall in recent years, so you hear the Agile founders saying it "failed." No it hasn't! The problem is the limitations of the systems implementing it.

You were an early signatory to the Agile Manifesto. Why was that document important? Why does it still matter?

The purpose of the meeting at Snowbird wasn't to write a manifesto. It emerged from the meeting. The principles were not even clarified until afterward, but they do capture the essential qualities of Agile.

I go back to the Manifesto when Agile isn't working. Which principle did we forget about? The Manifesto provides guidance.

Sometimes, as a coaching exercise, I'll ask, "How would you rewrite each principle for your purpose?" Take "Deliver working software frequently." What does that really mean? It might involve getting feedback from the people who pay money for the software because they define what "working" means.

What are the values of Agile software developers? What is the nature of the relationships it should create?

In our training, we ask people to go back and remember great team projects they were on – maybe in sports, a fraternity, church, or the military. Then they come up with keywords or phrases about their experience.

I hear "fun," "accomplished," "useful," "trust," "inspired," and similar words. Maybe ten percent of the time, people say, "productive." Most companies want Agile to become more productive, but people don't care about being productive at something they don't care about!

The people who do Agile well are focused. When they're there, they are really there. They're not available to respond to every twitch from universe. People become more productive, but if you go after productivity first, it doesn't work. Productivity is an emergent result of good team dynamics.

The Agile mindset is about recognizing when you did something and it didn't work, AND then doing something to improve it.

Toyota figured this out during the rise of lean manufacturing. They would open their doors to people from American car companies. The Americans thought Toyota was hiding something, which they were – they were hiding the team dynamics in plain sight.

Every time Toyota teams experimented, they wrote it up, and it had to fit on one piece of A3 paper. It covered what they did, why, how, and the results. They keep those A3 papers and review them before reinventing the wheel. Oh, we tried that and it did not work. But have conditions changed such that it would work now? They recognize what didn't work and why.

What is not talked about enough in the Agile world?

That Agile is hard. People have ways of doing something, and because Agile is a cultural change, they tend to go back to what worked when it gets difficult.

That tendency is built into our biology. We survive because our biology makes us good at repeating what works and avoiding what doesn't work (it's called evolution). And if you try something new, a lot of things won't work. You have to hang in there.

Also, management culture today is about having less people doing more work. You can use less people, but once the work piles up, projects can still end up with a 1,000 people. That's too many.

I once coached a project that had 30 people, and they weren't getting anything done. We picked seven people, got rid of everyone else, and suddenly started getting things done.

Jeff Sutherland has given the same advice for 20 years: managers are a central problem. Get the teams going first. Don't worry about anything else. Just get teams delivering software reliably and improving continually. Jeff does 'tough scrum,' which means if a team doesn't start delivering, he disbands the team. If the same person fails on two teams, he lets that person go. This stuff is not that hard, but you have to do it. Patience isn't talked about enough.

As a coach/consultant, how do you know when Agile is succeeding? What are the signs?

Basically, it's working if people are happy. As human beings, we don't like to work in malfunctioning systems. Jerry Weinberg, a 'grandfather' of Agile, said, "Remember, people are always doing the best they can in the system they're in."

If there are problems, usually it's the system. What structures or processes are inhibiting improvement?

If Agile is succeeding, the business should get better results. But they won't show up right away. There is a new culture, a new way of working, and new responsibilities. If managers are too short-term focused, they won't wait for the change, which can take three to five years.

Some people are just in the software industry because of the money. They won't make it through a long Agile transition.

AGILE PROJECT MANAGEMENT

Every technology company needs people who are too smart for their own good. cPrime has one named Kevin H. Thompson. He has a PhD in Physics from Princeton and has spent more time pondering Agile than is healthy for a human being.

Kevin has an uncanny ability to deconstruct the parts of a greater whole, which is why we tapped his brain for Agile Project Management. If you hate the term "project management," calm down. You know anger leads to the Dark Side of the Force (because you're in IT).

Usually, we reduce project management to frameworks like Scrum and Kanban. The question is, what do they all have in common?

Decisions, says Kevin, are the outputs of Agile frameworks. Agile should produce fast, quality decisions so that teams can spend most of their time creating code, hardware, etc. for projects, which are temporary endeavors to create something. Towards that end, Agile establishes a system of governance so that self-organization doesn't become *Lord of the Flies*.

In terms of sexiness, the word "governance" ranks somewhere between "compliance" and "org chart." But governance is essential to self-rule and the demolition of a hierarchy in which middle managers routinely rain hell down on product development. Governance keeps Agile running at three levels – Team, Program, and Portfolio – and has five components:

Roles: Agile frameworks set checks and balances by dividing power and authority among team members. This practice clarifies who

knows what and who can call which shots. Roles kill "rule by committee," which produces inane debate and little action.

Ceremonies. These recurring meetings with standardized agendas, attendance, and practices facilitate information-sharing and decision-making.

Artifacts: Artifacts are deliverables. The list of artifacts – and who completes which one – is a series of decisions. Artifacts make work measurable.

Tracking and Metrics: They keep teams honest, flag problems early, and sometimes lead to changes in the plan. Metrics tell who's getting what done and how quickly.

Governance Points: The moments of decision, which reflect the results of parts 1 to 4.

Today, most teams practice either Scrum or Kanban. The reasons for choosing one over another tell us a lot of *why* Agile matters at the project level.

crum is Adaptive meaning it's good for high uncertainty, and new products are uncertain. If you wanted to invent a drone that geolocate friends, flies to them, and delivers a verbal insult (a la Bowerick Wowbagger from *The Hitchhiker's Guide to the Galaxy*), it would be hard to foresee all the requirements. Perhaps you make detailed plans for two-week sprints and 'high-level' (i.e. vague) plans for the longer-term. Scrum seeks to maximize value delivered over time, but it can't eliminate the disruption and cost of unforeseen changes.

Kanban is a Reactive process, best suited for work that can't be planned. It's ideal for customer support, firefighting units, emergency rooms, and marketing where no request is predictable. It's for short-term,

high-priority projects. Although it's Agile, we wouldn't recommend using it for product development.

The point is that Agile Project Management reflects the *style* of decision-making a team will practice. Waterfall was designed for certainty, which we all learned is impossible in development. We build Agile on governance so that the HiPPOs (Highest Paid Person's Opinions) and PUBs (People's Unintelligible Buzzwords) can't hijack projects.

WHAT'S THE ONE THING YOU'D DO DIFFERENTLY?

Ken Olofsen - Runnable

Conventional wisdom says that Agile and its trunk load of lingo are just for software developers. Ken Olofsen, who has served as Co-CEO of Runnable and Head of Portfolio and Customer Marketing at Atlassian, says otherwise.

As Agile was becoming a 'big deal' around 2008, Ken joined the marketing team at Atlassian. He started using Agile products to manage marketers and was probably one of the first to do so.

Ken sees Agile as a culture of self-reflection. In this Q&A, he shares some smart questions you should steal.

How did you get into Agile and DevOps? What was the 'aha' moment?

I started my career as a software consultant, transitioned into marketing, and ended up at Atlassian in 2008. The 'aha' moment came from a company we acquired called GreenHopper. Back in the day, you used a sharpie, post-it notes, and whiteboards to do Agile planning. GreenHopper gave you

digital sticky notes and turned Atlassian JIRA into a virtual, drag-and-drop whiteboard.

Our timing was perfect. From talking to customers, we knew there was a desire to take Agile from the physical world to the digital. Agile was gaining momentum because it provided faster feedback through iterative development using weekly or biweekly sprints that produced functioning products. At the time, it was revolutionary to make something you could interact with in such a short timeframe. The enthusiasm for digitizing Agile led to JIRA Agile – now part of JIRA Software – and the introduction of other mainstream Agile tools.

How did your job change?

As the JIRA marketing guy, I hadn't written code in 15 years, but I was Scrum-certified and figured out how to develop an Agile cadence for marketing teams. We used biweekly sprints to prioritize work and commit to iterations. Eventually, we switched to Kanban because the nature of marketing is different than development. We had to react quicker to the business and real-world events.

Kanban worked in marketing because it forced us to reprioritize constantly. We had a finite capacity, so before taking on a new project, we had to justify that it was more important than what was already in the queue. You can't keep adding, so Kanban forced us to see our priorities through a zero-sum lens.

It's not always easier to say no to the new thing. Agile provided the framework for hard decisions.

What is the biggest avoidable mistake you see in Agile teams?

The biggest mistake I've seen is getting semi-educated on Agile – learning about standups, retrospectives, using a board to put tasks together, etc. – and latching onto ceremonies but not really understanding the philosophy behind them.

People get fixated on doing Agile 'the right way,' which we dismissed at Atlassian. There is no right way. The notion of Agile is to be self-reflective and see if your processes work. If yes, continue. If not, change it up. Don't do Agile a certain way just because someone said to do it that way.

Agile is evolutionary. You find a new technique, try it, and it fixes one problem but introduces another. You fix that problem and create another problem, and so on. If you adopt an Agile philosophy, you're committing to always challenge yourself and how you do things. It will always change.

Why do the "business" and "IT" usually have such a dysfunctional relationship?

The business has expectations about features, functionality, and timeline. If the business does a poor job of communicating what it wants, IT inevitably will make assumptions. IT works with constraints around budgets, time, and resources that the business isn't unaware of.

Problems arise when the business fails to set expectations, and IT fails to communicate its constraints. The relationship may vary depending on the type of IT organization you run. Some organizations think of IT as a cost center, primarily focused on keeping the lights on at minimal expense. Others try to push the envelope with innovation.

It's safe to say that majority of dysfunction comes from cost center IT departments. There's not enough investment and encouragement to make IT innovative and challenge individuals to do cool things. They'll always fall back on a constraint.

What are the most important questions you ask or get asked regularly?

When I was Atlassian, a new CFO came in and asked everyone the same question: **What's the one thing you'd do differently?** It was a fascinating psych experiment. Effectively, it put people on the spot to think, if I had a magic wand, what would I do?

Sometimes, multiple people or teams gave the same answer, and that was telling. It was a sign of concern. **It's often more difficult to say, "That's a bad idea," than to go along with precedent, even if the way we've been doing it forever doesn't make sense.**

That's why Atlassian talks about Agile and DevOps as mindsets rather than techniques. They encourage everyone to say, keep doing hard work, but have time and space to ask if it's the right use of effort.

Every two weeks at Runnable, we did a retrospective to discuss what did and didn't work. Maybe we had a failure or outage, but we were not trying to 'out' anybody. Rather, we'd ask, **"How were you in a position to make that mistake without someone else stopping you? Do we need a checklist? What would prevent that failure in the future?"**

What's the difference between Agile and DevOps?

While Agile and DevOps focus on different areas of the software development process, they're basically the same concept. The belief in Agile is that

at the end of every sprint, developers should have a functioning product to share with their team and customers. DevOps' role in the story is making sure the code we write is deployed onto some machine and working as quickly as possible.

Agile brought down release cycles from months to two or four weeks. DevOps brought us to daily and nightly cycles. The next step is to commit that last line of code and deploy almost instantly.

AGILE PROGRAM MANAGEMENT

Agile at the team level is cozy and almost tribal. After a two-week sprint, you could put on a blindfold and tell your coworkers apart by the smell of their lunch. But what teams receive from above can entirely undermine the projects. If your plumbing sends water through rusty pipes to a clean glass, you still get a glass of rust-filled water. Same with Agile – when bull-shit requirements travel from top to team, it contaminates project manage-ment. That's why Agile teams need Agile Program Management.

The program level addresses questions like, who should be on what team? How do you develop requirements for multiple teams? How do you plan and track work across teams doing distinctly different projects? How do you make sure each sprint serves the strategy of the business?
The program managers coordinate multiple projects to achieve the buzz-word you've been craving for five pages: Alignment, puh (we spit after writing it).

Alignment (puh) is among the top PUBs (People's Unintelligible Buzzwords) of 2017. It's great for making people think you agree on objectives over which you had no choice. Most business stock photos depict alignment.

Real alignment is a state in which people, processes, and technology work towards accomplishing the same thing. Like the mythical Unicorn, Align-ment tends to soak up a lot of money yet never reach its lofty aims. Why? Because there's no program management layer.

Program management is where the three governance levels -- Project, Program, and Portfolio – start to matter. Program is the bridge, the unifier, the cheesy protagonist in a movie that says, "We're all in this together" after working out everyone's interpersonal baggage.

The question of program management is, how do we get multiple teams to produce the pieces of release that I can send to production? The trouble is that projects tend to be interdependent. Again, if I want to make the Insult Drone, how would one team begin to write the insult personalization scripts without first having the facial recognition code? Three people ensure that projects don't run into such issues:

Team Product Owner
Controls product requirements for up to three teams.

Area Product Owner
Controls big-picture product requirements, decides on content of releases.

Program Manager
Deals with collaboration among teams, including addressing dependencies.

Their North Star is usually a Burn-Up chart that plots the scope of a release and work actually completed. It shows whether work is happening at a steady and predictable rate. If not, why? Is one team code-jamming another due to a dependency? Did one set of requirements overload a team?

Essentially, the Burn-Up captures the purpose of program management: to ensure that releases go to production on time and contain what the bigwigs expect them to contain. It's distinct from project management in that the program roles manage teams, not individuals.

If program management sounds less concrete than project management, that's because it is. As Agile scales, the distance between the code written and people defining the requirements expands. For a visual, imagine that you're in army platoon out on a field next to friendly platoons. The general is up in a balloon with a microphone, and he's yelling out orders in Latin instead of English. You pick out a few roots words, but really, you have no idea what the general wants. The program management layer translates for the platoons.

IF I GET HIT BY A BUS...

Nirav Parikh - Redbox

Some people are so damn rigorous in their Agile practice that their teams seem to move gracefully, like ballet dancers or frolicking antelopes. Nirav Parikh, VP of Technology at Redbox, is way too modest to ever claim such prowess, but it's true – he makes Agile sound and look good.

Nirav is an alumnus of DirecTV, where a lot of great Agile and DevOps happened in the last decade. In this Q&A, Nirav drills into Agile details that will make you reflect on how you solve problems, define success, and stay on course. How resilient are you against turnover, sickness, or getting hit by a bus?

When did you realize you had to change how you do your job?

In 2004, when I was at DirecTV, we started looking for a better way to develop, and by 2007, we had implemented Scrum. The problem for us was that when you solve complex problems, you can't think through everything first. Imagine a puzzle. You start solving it and along the path you find solutions. You could never define the whole puzzle before you begin.

We had reached a point where it took too long to put anything together. The business was trying to set requirements without knowing what technology was capable of doing. With Agile, we could work in smaller increments and change along the way.

How did your work change after going Agile?

Agile is a phenomenal transformation for folks on a team. But if you're in a traditional organization in a leadership role, the transition is harder.

I was on the fence being part of a team but also a leader. Initially, it was complex. How would I manage and lead work for three different agile teams? At the time, there weren't enough tools to aggregate all the information together, so we started with physical whiteboards.

Agile was a powerful way to estimate, plan, and understand work. Ok, I've got seven stories with 25 points. One team probably can't do it in one sprint, so I'll chunk it up. Estimations and longer-term planning became easier too. One epic becomes seven to eight sprints worth of work. If you have one quarter, sure, it's doable. In one month, probably not.

Whiteboard Agile struggled when you hit 300 or 400 developers but became easier in 2010 when tools started to aggregate info. Today, I can see one dashboard with everything I need.

What's the most important question you ask your teams regularly?

At our Scrum of Scrums meeting every other day, I ask, "Where are the blockers?"

Even in a big organization, it's a powerful way to understand what's relevant and important now. The question solves roadblocks in a quick, efficient manner.

What's the most crucial part of software development?

I think it's having an organization where you're not thinking in individual contributions but rather in terms of alignment and a shared belief in how we do things and make decisions. If you're aligned on decision-making, you'll be successful.

Scrum is certainly good for accomplishing that. There is a common vocabulary with a definition of what we call "done." We agree that we're going to build a new company or product with a certain purpose. If we can align on that, then we can decide what to do for the next two weeks.

What does it mean for Business and IT to "get" each other?

It means you measure results the same way. KPIs [key performance indicators] are so powerful because you agree on what you're trying to accomplish. When we base decisions purely on time and cost, we disregard that in business, tech is just one part of the solution.

For example, conversion is a great shared KPI. In a web app, we're trying to get certain numbers of conversions from step A to B, B to C, and so on before a person leaves the app. In the tech organization, a sub-KPI may be performance. How fast does the app load? How fast do interactions happen? If the next page doesn't load in under a second, it could hurt conversions.

But again, performance is one of many variables in a large equation. If performance leads to better experiences that generate conversions, great. But we have to agree on high-level KPIs before figuring out which sub-KPIs will support them.

What is the biggest avoidable mistake you have made in Agile?

The biggest avoidable mistake is thinking you can keep some rituals and discard others. It always backfires. Standard rituals might include backlog grooming, sprint planning, daily standups, retrospectives, and plan fest [quarterly planning where you break up stories]. Choosing to avoid or skip one backfires so quickly.

At Redbox, we thought we could get by without backlog grooming, and sure enough, we had a sprint or two where on day one, we discovered we weren't ready. A whole team didn't have enough to work with. In an environment with multiple agile teams working towards large releases, that's a problem.

The beauty of Agile is the fact that you do things in small increments, and each piece has distinct value. It actually takes more planning, more discipline, and more structure than traditional methods.

Waterfall has a distinct planning phase that could last two months. In Agile, there is quarterly, monthly, bi-weekly, and daily rituals. Teams may not spend as much time on planning phases, but there is more rigor, information processing, and quality in Agile planning.

Forget what keeps you awake at night. What helps you sleep?

What I worry about least is that if Agile is done right, my teams are self-sustaining. If I get hit by a bus tomorrow morning, or someone leaves, I know someone else will step in.

Agile produces self-sustaining teams if it's done right. The realities of life mean you can't be dependent on any one person. You have superstars, but there's always someone willing to step in.

AGILE PORTFOLIO MANAGEMENT

In Kurt Vonnegut's novel Cat's Cradle, a U.S. marine general asks Felix Hoenikker, the co-inventor of the atomic bomb, to invent something that will help soldiers cross swamps. The child-like scientist, unconcerned with the repercussions, creates ice-nine, a form of water that freezes at room temperature. Ice-nine has the unique ability to retrain the molecules in any body of water to freeze at room temperature too. It's fine for a swamp crossing if you're willing to freeze every natural body of water in the world, as they're all interconnected.

Agile without portfolio management is a Cat's Cradle science experiment. The invention probably won't destroy the world, but it might fail to achieve any discernible purpose, or worse, inflict unintended consequences on a company.

Agile Portfolio Management is about making sound investment decisions for the organization. It sets the goals that determine releases, which determine the nature of projects. When technologists lament the IT-business divide, usually their woes stem from failures at the portfolio level. Waterfall portfolio management also defines, evaluates, plans, executes, and tracks major business initiatives. The difference in Agile Portfolio Management is that it permits adaptation when the needs of the business change. Whereas Waterfall assumed that the agenda delivered from the boardroom would translate flawlessly into a final product, Agile embraces the chaos that is development.

The driving question of Portfolio Management is "Why?" In Cat's Cradle, Hoenikker is a one-man Agile team with poor portfolio management. The general defined the requirement – something that would enable his troops to cross swamps – but his requirement didn't capture the business case: Why do his soldiers need to cross swamps in the first place? What are the benefits of crossing swamps? (Not causing Armageddon should be an obvious requirement, but then again, Cat's Cradle is commentary on nuclear technology, which can destroy the world.)

"Why" is usually the first casualty in product development. The upper management, speaking in juicy PUBs ("customer experience"...blah blah), delivers requirements that are hard to understand in the language of development. It's the old joke where two men ask a genie with a hearing problem for wishes. One ends up with a million ducks and the other gets a 10-inch pianist.

Frustrated that IT doesn't get it, the upper management turn to governance – more reports, meetings, and other burdens that deflate the tires of Agile. To Teams and Program leaders, too much governance from above is an imposition. Over-governance, you'll recall, is what the Agile Manifesto revolted against.

So, alignment is not just about the achievement of goals, but integrity in how teams function. Agile teams divide up tasks and do their work however they choose. Uppercase "Agile" companies trust that such teams will perform faster and better than teams micromanaged by a bitter middle manager whose real dream was to raise a best-in-show poodle named Henley. When the portfolio level tries to force alignment that way, it backfires by jamming up the workflows that align people to teams, teams to programs, and programs to portfolios.

The new role at this level is the Portfolio Manager, the Marine general. This person approves, rejects, or schedules initiatives based on business cases

presented by the Area Product Owners and Program Managers. The key is to have that duo in the Project, Program, and Portfolio discussions. It's their responsibility to say, "General, the teams said they could invent a substance capable of freezing all water on earth. Are you sure that's what you want?

SPECIFY THE 'WHAT' AND LET TEAMS FIGURE OUT THE 'HOW'

Ken France - Blue Agility

Since the Manifesto's debut, Agile has evolved from a tribal, team-scale activity to a philosophy that runs Fortune 500 companies. When multibillion-dollar enterprises want help with Agile transformations, many turn to Ken France, founder and CEO of Blue Agility.

In this Q&A, Ken shares why Agile bombs if the top executives at a company don't buy into it. One of the problems, says Ken, is the never-ending, ever-proliferating list of cockamamie Agile definitions on which no one agrees.

We're with you, Ken. Read on for some Agile and DevOps wisdom that rises above the terminology fray.

Tell us about your role in the Agile industry.

My company Blue Agility focuses on large-scale Agile transformations. I started my career as a developer, moved up in the ranks, and have now spent most of my career in consulting. Besides running Blue Agility, I help our clients at the executive level.

How did you get into Agile and DevOps?

My first job out of college was at an air traffic control system. They were in Waterfall mode. Right when I started, they got a letter from the government saying our company wasn't meeting milestones and our contract would be canceled if we didn't step it up.

We moved towards something we called "iterative" or "incremental" development back then. We even integrated automated builds, which you'd now classify as DevOps.

With iterative development, we could break problems into smaller pieces and build parts of solutions while testing and implementing others. The concept of getting something "working" and "out the door" changed.

Initially, we worked on large project teams rather than cross-functional Agile teams. But the evolution towards smaller teams changed our thinking. We began to specify the "what" and let teams figure out the "how."

What is the biggest avoidable mistake you see in Agile?

The biggest mistake you can avoid is having Agile be and remain a grassroots effort in the development side of the house. Leaders say sure, do it. **If it's not a whole organization endeavor, you'll have some short-term success then go back to your old ways of working. Inertia will win out.**

You have to scale to the whole organization if you want Agile to work, but you could have people report up the chain. Organizational boundaries become more about performance evaluations and paychecks than how works get done. If some team members report up into development versus support versus IT, it doesn't matter as long as they work as a team.

What do Agile practitioners disagree about most aggressively?

There's a lack of understanding about Agile and DevOps. How do you define each one?

Agile is usually equated with certain practices, like Scrum. People don't go back to the Manifesto. It's all about technique now. DevOps is usually described in terms of CI/CD and tools, not the aspect of bringing development and operations together. The biggest disagreements come from defining each one and the overlaps.

I'd say Agile is a prerequisite for DevOps, but in order to truly be Agile, you need to adopt aspects of DevOps. I've been using a Venn diagram analogy that seems to resonate with people. Some things may be uniquely Agile, and other things are uniquely DevOps, but the overlap in the middle is so large it doesn't make sense to separate them.

What's the most important question you ask clients regularly?

I ask leaders to think about how their clients view the services they get. The focus is often on internal mechanics. Agile is equally about clients. I ask, "What would your client say? Are they getting value from you?" Putting people in their clients' shoes creates the right mindset.

You stress the importance of portfolio management. Why does it matter?

Ultimately, if companies want to achieve better outcomes, they must set direction and route work in a better way. There are some symptoms when companies are failing in those areas. I'll ask, how many projects do you

approve each year? 200. Ok, how many get done? 50.

I can tell executives how to do a better job of prioritizing and setting direction, but they must trust teams to get the job done. It's a mindset change. Trust that folks know best how to achieve their goals. They just need to know what the goals are and their priority.

Are Agile and DevOps fads?

I think they'll endure. **Different techniques and frameworks will arise, but the underlying principles are not fads.** Breaking problems down into small, easy parts is just a better way to work. These techniques are here to stay, but frameworks and words will change.

CONTINUOUS INTEGRATION / CONTINUOUS DELIVERY (CI/CD)

Picture Dev and Ops, a couple *so* dysfunctional that MTV has decided to make a reality TV show about them. They call it *Contested Development*. In Season I, "Waterfalling," Dev and Ops sit down for those stream-of-consciousness, he-said-she-said interviews. Their monologues go like this:

Dev: Ops is a total code-blocker. I come up with all this brilliant stuff, and Ops is like, "Err, excuse me, have you reviewed that with the change management board?" Total buzzkill. Why am I married to this killjoy?

Ops: Whah-ever! Dev goes out drinking with friends, comes home and is all like, "Uh, I got an idea! Let's write unstable code that doesn't work on the servers. Dev is the worst."

By the end of Waterfalling, Dev and Ops sleep in separate beds and wage psychological warfare. If you think this is a hyperbolic description of the Dev and Ops divide, we beg to differ. Brandon once walked into an interview for a product development position, and the *interviewer* broke down crying because the tension between Dev and Ops had gotten that bitter.

We're opening a chapter on CI/CD this way to make one point loud and clear: **DevOps is *not* CI/CD.**

Soak that in.

Our DevOps interviewees agree on almost nothing except that a) No one has the same definition of DevOps and b) DevOps is the union of Development and Operations into a single organization (plus quality assurance or testing if they aren't included in Dev). CI/CD matters, in part, because it breaks the Dev and Ops standoff.

If Agile is about making better decisions faster, CI/CD is about making Agile sustainable. Organizations lose faith in Agile because it's impossible to keep up without DevOps automation. CI/CD is the core of that automation. If you will, bear with us on the definitions. We need to set these straight. **Continuous Integration** is the process in which developers and testers regularly merge and validate new code. In the past, developers wrote code and integrated it once a month for testing. It was silly because flawed code from four weeks ago forced developers to revise code written one week ago. So, CI automates the integration and testing so it can happen continuously.

Continuous Delivery is the process of continuously creating releasable artifacts. The teams automate code building and testing so they can work in short cycles. CD teams can release to end-users once per day or even multiple times daily.

Put simply, CI/CD is a process for continuous development, testing, and delivery of new code.[1]

You can see how CI/CD gives Dev and Ops a chance to rekindle their relationship. CI is for Dev, who otherwise might send complete duds to Ops. CD is for Ops who, rightfully suspicious of Dev, might stall a release and ask annoying questions to be sure the code won't blow everything up. CI/CD matters because it enables Dev and Ops to work together **by trusting**

1 CI/CD gets mixed up with **Continuous Deployment**, which is the concept that every change made in the code base will be deployed immediately to production. Yes, that's terrifying and unnecessary.

a process, not each other. While that sounds negative, think about how you pay for goods at the grocery store. You trust the processes established by banks and credit card companies, *not* the cashier who swipes the card or the company he works for. Trustless systems eliminate situations that lead to drama.

Of course, CI/CD has less lofty benefits. It enables you to find and fix problems faster, automate lengthy manual processes, and release more often. If you release more often, you beat competitors to market with a higher quality product and make more money than them.

We know, "money" can seem like a dirty, politically incorrect word. That's why businesspeople cloak it with terms like "revenue," "profit," the "the bottom line," and other jargon that doesn't make you think of cloth paper with pictures of dead presidents.

Look, Dev and Ops married each other for the money, not because they were both seeking their "best friend" who watches all the same Netflix shows and "loves travel." No, this was an arranged marriage.
Even though DevOps is a marriage of convenience, we can't eliminate the human factor from CI/CD. It takes months to set up and requires all sorts of people – quality assurance, Ops engineers, Scrum masters, etc. – to bury the hatchet. You can automate up the wazoo, but that doesn't mean Dev and Ops will automatically hold hands and skip into the sunset.

That's because DevOps is a culture, and CI/CD is an institution of that culture. Likewise, Americanism is a culture, and trashy reality shows are an institution of that culture. Institutions are the rules of the game that provide some payoff, tangible or intangible, to players. DevOps teams get speed and quality out of CI/CD, and bored couples get exhilaration and faux purpose from reality TV drama.

Again, we must emphasize: DevOps is a marriage of convenience, and CI/CD helps the couple meet business goals. Technology executives know that continuous evolution, quick fixes, and quality results create and keep customers. Likewise, they know too well that one botched release can rack up enough angry App Store reviews to tank a product.

If Season II of *Contested Development* is "DevOps," Season III is when DevOps makes a shizzle-ton of money. Then, the company spends it on quinoa bowls, in-house Shamans, and reclaimed barn wood (which some farmer is laughing about in his new, scrap-wood-funded barn). A tip of the hat to CI/CD for funding all three.

DONE DONE, OR DONE?

Cameron Deatsch - Atlassian

Marketing Agile software to companies is challenging because, as you know by now, few people agree on what Agile and DevOps are. That's one of many reasons why we respect Cameron Deatsch, Head of Server and Enterprise Marketing at Atlassian.

Cameron works to provide clarity amongst the confusion, specifically for organizations attempting to adopt these practices at scale. As big Atlassian fans and partners, we know that he is unusually successful at that task. As Cameron shares in this Q&A, the iterative nature of Agile is foreign to most enterprise execs. It can lead to inspiring successes – or confusion and hilarity.

You use Agile on your marketing team at Atlassian [read Ken Olofsen's interview for some background on this]? How did your work change after going Agile?

I think we get more done, but it's hard to measure. We look more chaotic from the outside compared to when we weren't Agile. With teams debating back-and-forth, you'd think there's no way in hell we get anything done. **Making decisions collaboratively can be an ugly exercise, but usually, it arrives at the right solution.** Rarely do my teams need me to make a final decision, but maybe they just want to avoid me.

What is the best example you've seen of DevOps in motion?

Atlassian acquired Bitbucket in 2010. When I joined Atlassian in 2012, Bitbucket was a relatively small team serving less than one million users. Soon, we quadrupled that number, and the millions of extra users strained the team and infrastructure. Everything got overloaded. The effect on performance was very public and had to be fixed.

We brought in new leaders who implemented DevOps. They put in a variety of processes that shortened the time between identifying a problem and doing something about it. When Bitbucket went down, we used to hear about it from customers. How could we get monitoring in place to prevent that? How could we relay feedback back to product teams to prevent the next outage? How could the team develop iteratively yet make sure nothing broke when changes went live? The new leadership sorted that all out.

Thanks to their changes, Bitbucket has virtually no outages now, and the team can roll things out and back quickly if there are issues. They frequently launch 'dark features' [features visible only to a small number of customers] before making them public. The team could deploy three to four times per day if they needed, but generally they do one release daily.

Talk to us about Agile and DevOps technology. What are the biggest misconceptions about it?

The biggest misconception is that it will solve all your problems. Agile is a way to plan, iterate, and execute better products. It does not mean your strategy is sound. If you have a good product strategy, Agile will help you get there faster with a higher quality result. It's just a practice.

Agile is not the band-aid you put on a dysfunctional organization or leadership team. Oh, the developers aren't happy, not working well

together, and have no clear direction? Great, let's be more Agile and all have birthday cake. Doesn't work that way.

If you have good leaders in place who trust and respect one another, Agile can help them all work together. If you don't have a culture of trust and open dialogue, standups won't be pretty.

Team playbooks, Scrum, sticky notes, etc. are nice, but people need to trust each other and agree on goals, otherwise how you do planning doesn't matter – it will fail.

Why do business executives struggle with Agile culture?

My favorite Agile quote of all time comes from a banking executive. During a visit with Atlassian, he said, "So, this came up recently where I was told something was done, I told the rest of the leadership team, but it wasn't done done. It was Agile done. How do I know what's done done, or done?"

We see this situation all the time where executives don't know how Agile iteration works. This banking exec lost face by claiming something was "done" when it wasn't done in his sense of the word. He wasn't blaming us; he was just confused.

The problem in Agile is that you're never done. You're constantly iterating and could have four or five parallel experiences running at any given time. When you plan to announce a major new feature, it's not ever done, but it needs a certain level of quality and availability before you tell all your customers. That announcement is the closest thing to "done done."

What challenges does your organization have with helping companies become Agile?

They disagree with how strict of an Agile methodology belongs in products like JIRA. Atlassian products have a loose approach because there are so many different forms of Agile. Honestly, that's our number one battle. People want their method baked into the product, but that would restrict the product, and methods are always changing. We don't want to tell the market how it should do Agile.

Is DevOps a fad?

More and more of our customers have changed their titles to "DevOps Manager" or "DevOps Architect." When people start changing their roles, that tells me they think DevOps is real. They are being paid to have that title.

INFRASTRUTURE-AS-A-SERVICE (IAAS)

Dev and Ops, reunited by their love of CI/CD during Season II of Contested Development, get a little cozier. Dev comes home (sober) one night with red roses and Rioja and says, "How about we chill and watch Netflix?" Ops, missing the cue, actually turns on Netflix. The couple joins the millions of viewers who cause Netflix to consume 37 percent of North American Internet bandwidth during peak hours.

Their evening is possible thanks to Infrastructure-as-a-Service (IaaS), a phrase you never use, imagine, or remotely think about when you roll in with roses and red wine. In the simplest terms, we're talking about a dynamic cloud infrastructure that adds computing resources whenever you need them. It serves DevOps teams on both the development and production sides.

Why does IaaS matter? To answer, let's begin with the story of the cloud. In the early 2000s, companies learned that owning datacenters is painful and expensive. Sniffing opportunity, specialists built datacenters and rented out portions to hundreds and then thousands of different companies. They called it the "cloud" because ruining the majestic beauty of waterfalls wasn't enough for the technology industry. Then, they came up with Infrastructure-as-a-Service, IaaS. Is it pronounced eye-azz, aye-yes, or eye-ass? No one knows.

Cloud infrastructure matters because traditionally, creating new environments was miserable. Companies relied on golden copies, one-off scripts,

and manual configuration, which incinerated expensive labor hours. It also made CI/CD impractical because configuring test environments created bottlenecks, especially with, say, 300 developers trying to test and release daily.

On the operations side, a lack of cloud infrastructure can be fatal. When Dev and Ops turn on House of Cards for their date night (and try to ignore its semblance to reality), they and millions of other Americans don't suffer buffering. Netflix's cloud infrastructure can expand capacity as needed.

Amazon Web Services (AWS), Netflix's cloud of choice, is like the lawn of a global concert venue. It would be too expensive to provide everyone with seats, and you might not fill them each night, so the venue takes over as much or as little of the lawn as it needs.

Like all DevOps practices, cloud infrastructure is about money. When a team needs additional servers, they're available without delay. And when a team doesn't need them, the company doesn't pay.

Our interviewees agree that the future of IaaS is containerization, so let's touch on it. If server instances are like toothpaste squeeze tubes, containers are the technology that gets out that stubborn last bit you'd normally throw in the garbage. It adds to a lot of toothpaste over time. Whereas traditional virtual machines (VMs) run storage-hogging copies of the operating system and libraries, containers use a shared OS instance. In non-geek speak, you get more resources from the same servers (ok, still geeky).

Behind the scenes of DevOps, VMs and now containers ebb and flow so that couples like Dev and Ops can binge-watch Netflix shows without buffering that might, heaven forbid, force them to talk. On behalf of the tech industry, you're welcome for that heightened state of romance.

I HAD NOTHING LEFT TO DO

Roshan Duraisamy

17 years ago, Roshan Duraisamy introduced CI/CD practices at Symantec in New Zealand. The concept of DevOps didn't exist yet. If you've ever used the phrase "ghosting a machine," thank Roshan and his team. They developed Symantec Ghost, which, for a while, was one of the only ways to clone a machine to thousands of others.

Today, Roshan is the Senior Director of Engineering at MobileIron. Tactical and obsessed with quality, Roshan is the consummate DevOps guy and has much to offer in this interview:

How do you define DevOps? Why does it matter?

In my career, I've seen companies interpret DevOps to be the CI/CD space. Instead, I view CI/CD as a framework that enables DevOps as a mindset and practice by the team. In my definition, DevOps eliminates the distinction between building and operating. If you develop it, you operate it too.

In the most optimal DevOps strategy I've seen, one integrated team owns building, testing, support, and operations. They divide teams not by function, but rather by release.

Everyone wants to make the product as close to right and perfect as possible when they know they'll be doing customer support. Who

wouldn't want to build quality into a product if that were the approach?

How did you get into DevOps?

My first CI/CD job was with Symantec in New Zealand. The first change was to automate the builds and then integrate tests. I was the Test-Driven Development Lead and junior-most engineer. Management asked me to evangelize test-driven development to the whole team. It became my job to ensure that no line of production code was written without a test attached to it.

It was challenging. Folks had been doing software development for decades, and here's this junior engineer trying to sell them a different way.

When I joined that team, we postponed release dates by months sometimes. After the transformation, we didn't miss a single ship date for years. Automated CI/CD principles made a huge difference for delivering on time with high quality. Afterward, folks said they could no longer imagine what life was like without automated testing.

What is the biggest avoidable mistake you have made in DevOps?

The biggest mistake I've made is not learning enough about a product before building infrastructure for it. In the case I'm thinking of, I was supporting a product that had been around for 30+ years. It had to be tested and used a certain way, but we wanted to get the biggest bang for the buck, so we built our system for the most common use case across multiple teams.

The product required specific, static IP addresses with no NAT [network address translation], but we couldn't provision them the way the we set up the infrastructure.

The product team had heard all about the great things we did for other teams but couldn't use what we built. **The lesson was that if you do DevOps for multiple teams, you have to understand each one correctly. You can't use a one-size-fits-all approach for every team and product.** If I did that project over again, I'd build stronger relationships with the team before trying to create infrastructure.

What's the most important question you ask your team regularly?

What did you have to do by hand, and how often did you have to do it? It's a filter on the level of CI/CD. **The best indicator of a successful practice is that the CI/CD engineer has nothing to do.**

What's the most crucial part of software development?

Most crucial part of software development is quality. A product may solve use cases once you understand them. But without a high level of quality, the product will tank.

One way to measure quality is by looking at how often customers call into support or access support articles after a release. ServiceNow does that. They measure all knowledge base, forum, and support hits and aggregate that data for the scrum team responsible.

In the successful products I have been part of, we made quality our top priority, deadline second, and features third. Startups tend to put features or deadline first and quality third. So, they run into problems.

If you want renewals and more customers, quality comes first. If you want quality, don't write production code unless you can test it. It's better to deal with quality issues in-house than with customers.

Forget what keeps you up at night. What about DevOps helps you sleep at night?

On a team I inherited eight years ago, builds were automated but brittle. We overhauled the product code, building and testing scripts, networks, etc. at five different sites. It took two years to fix.

That kept me up at night – literally – because I would get calls about something that was broken. After the overhaul, we went from every other build breaking to completing projects without a single breakage. **It was time to switch jobs because I had nothing left to do. That made me sleep well.**

TEST AUTOMATION

In a moment, you're going to read an interview with Glenn Trattner, COO of Quantum Metric, who teaches his team to ask these questions: "What did you do today that was manual? What didn't you like doing? What task was most annoying? Figure out how to automate it."

That is the essence of test automation. Testing is crucial because if we screw up a release, we risk ruining Dev and Ops' date night. But no one enjoys testing. It is someone's most annoying task, highly error-prone, and therefore ripe for automation.

Testing became a shoot-us-in-the-face task as systems became more complex. It's not unusual to have ten different applications talking to each other in one B2B platform. Test automation happens right in CI/CD, which we've introduced already. It needs its own chapter because, in the early DevOps stages, automation can become screw-up central.

Why? Because new DevOps orgs tend to automate the wrong things, which, frankly, is a problem throughout tech culture. Case in point: Alfred. Despite all the serious problems available, those guys automated the process of finding butlers and telling them what to do. Sorry, we meant "Alfred Home Managers." Definitely not butlers. Definitely not stolen from Batman.

The question, "Does this need to be automated now?" gets a "yes" too frequently without some reflection. How often is the process repeated? How long does it take? What teams get delayed if you don't automate it? Is it an error-prone process?

The heart of Agile is decision-making, and choosing what to automate is a

tough decision. The reality is that people will push to automate their own most hated manual processes. At some point, that's fine. Great ideas come from frustration. But early in DevOps test automation, it pays to be discerning.

For example, most companies should automate functional testing before user-interface testing. They both take two to three hours, they both get repeated daily, and they both involve multiple dependencies. But if you automate functional testing, you don't have to update the automation scripts very often. Conversely, the UI requires frequent script changes, and no, DevOps teams do not have an Alfred who can handle that pesky task. Time *is* a luxury.

There's no such thing as over-automating. If you had unlimited people, time, and money, you'd automate everything possible. But we'd be the first to admit that total test automation *isn't* possible. You can break down tasks into smaller pieces and automate in patches, but not always. Sometimes, you just document the process in detail, execute it manually, and not enjoy it. But afterward, if you're *worthy*, you can offload your domestic chores on a poor soul who has to list "Alfred" as her job title. Ah, the Silicon Valley dream: to name your employees after a comic book butler.

Again, we can't avoid money, the M-word, with test automation. Yes, it can make your development cycles 20 percent more expensive, but that's chump change next to deploying a busted release that crushes your reputation. Fast iteration eventually blows up in your face without test automation.

By the way, "Blow up in your face," would be a great service from Alfred in those moments when you're too lazy to yell at your neighbors about their dog that always dumps on your lawn.

IT'S A BUZZWORD. I'M EXCITED FOR IT TO GO AWAY.

Glen Trattner - Quantum Metric

If you want to 'get' what the Agile Manifesto is all about, dig into this Q&A. Glenn Trattner, COO of Quantum Metric, personifies everything we love about the Manifesto. It's thanks to people like Glenn that "DevOps" is a thing. But as you'll see, he *really* hates that word.

Raw, direct, and irreverent, Glenn took the interview in directions we couldn't have anticipated.

Tell us about your background with DevOps

I've been in IT operations for 11 years at a media company [in addition to his role at Quantum Metric]. I help deploy apps to test and production, build better monitoring, improve availability, etc.

For the better part of a decade, whenever we've had an issue in deployment, a change, or some other issue, solving it has been a group effort among the development, testing, and operations organizations. We've run weekly ops reviews every Monday at 11 am for many years.

The idea of the meeting is, if we have an issue, we need to talk about it and

all learn. It's not just one team's responsibility. Through that method, we did our best work. There's no throwing anything over the fence. We have an ingrained culture where the production environment is everyone's problem.

So how do you define DevOps?

My definition of DevOps is development, testing, and applications services [a.k.a. IT operations] working in concert to deliver a business service. People say, "We implemented DevOps." You can't do that. What did you actually do? Seriously, ask people how they "implemented" DevOps, and they won't be able to answer.

Some companies will call a "development team" a "delivery organization," by which they mean development, testing, and maybe project management, but not ops, who are of course part of the delivery. You need all of them. I don't love using the word "DevOps." That fact is, it's a buzzword. I'm excited for it to go away.

I call my team "application services." Dev writes code, testers make sure it works and performs, and app services is everything else. Is the environment built and ready to run? Is there high availability? Can it scale? Is there monitoring in place? Do we know at 2 pm on Tuesday if what is happening is good or bad? Is the business functionality working as intended? We handle all that.

Why does DevOps matter?

If you compare the average mobile app to a call center app, there's a whole different world of complexity. Enterprise apps have fewer users but far more functionality, integrations, crazy unknowns, and permutations of what people can do.

CI/CD concepts are fun, and executives at big companies try to do them, but it's a bit silly to reduce DevOps to that. The whole idea of DevOps is continuous improvement in the delivery of apps, whereas CI/CD is just about automated coding and testing methodology.

Look, we've been doing DevOps for 15 years, but we don't want credit for being 'early adopters.' It's the only way to deliver if IT is supposed to be a business differentiator. Usually, DevOps is not 'the thing.' It just helps you deliver whatever 'the thing' is.

Even at an IT company, you're not the thing. You're trying not to be a cost center or pain in the ass to a business that wants things done but not for a bazillion dollars. The only way to do that is to work as a team.

It's super obvious but not the way IT traditionally works. There used to be silos, walls, and handoffs. **It's a waste of time to make walls.**

What's the most important question you ask your team regularly?

The biggest questions I ask are around what we learned and what we can do better. At our Monday meetings, we look at recent incidents to see what went wrong. What happened? What was the business impact? Why did it break? What could you have done better in resolving the incident? Could we have fixed it faster? What are we going to do differently to make sure it never happens again?

When a functional issue is found in production, my favorite question is, was that found in testing? The answer is no, or we wouldn't be having the conversation. Why wasn't it tested? Well, we didn't think to test that, or we

can't test that in a test environment? Why? Well, we don't have the data. Let's get it then.

A decade ago, we wanted to smoke test before releases went public. Someone said, well, we can't order something because we'd actually have to order it. How would we pay? What account would it go on?

Easy. We got real American Express cards, ordered stuff, and canceled it. What about reporting? We exclude those transactions from reports. You ask questions around how to improve continuously and problem solve as you go.

Another great question is, "What is your teachable point of view?

" I was once asked to think about it and make a video explaining my teachable point of view. What I came up with ties nicely back to DevOps.

At the media company, we needed to do things faster, better, and cheaper. In fact, we had a BHAG, a Big Hairy Audacious Goal, of doing everything 50 percent faster for 50 percent cheaper. How?

My answer was to ask some questions. What did you do today that was manual? What didn't you like doing? What task was most annoying? Figure out how to automate it. Tomorrow, you'll do something else that's annoying and that you never want to do again. Automate that too.

The goal is to work yourself out of a job, though that's not how it works in reality. The more you can automate, the better your environment will become. When you figure out how to not do what you find annoying, you make the whole process better. It frees you up to improve something else.

That was my teachable point of view. Find annoying things and make it so you never do them again. But people fear that someone will lose their job if DevOps automates processes. It's not true. I can't think of a time that a DevOps team wasn't growing over the years. "Thanks for automating everything, we don't need three of you," has never happened and likely never will. A robot in an assembly line is another story. It takes a job and gives it to a developer or engineer.

In your career, what Agile experiments have gone surprisingly badly or surprisingly well?

We had a concept called F-12, which was a 12-step, AA-style program to get over fear of failure. People disliked it at first, but over time, it became extremely successful.

The idea was that you don't want to encourage people to take unreasonable risk that will inevitably cause a catastrophe. But, you do want people to take calculated risks that could fail but might make something better. The idea of F-12 was not to fear repercussions if you did fail. We didn't want leaders to make an example out of somebody because we had a production outage, even if it was a human error.

If people are always afraid to do something, they won't do anything, good or bad. There's no improvement in that atmosphere.

Along the way with F-12, there was a production outage or two and cases where leaders promoting F-12 didn't live the message. No one got fired, but public lashings in project meetings definitely happened when someone tried to do the right thing and accidentally screwed up.

In the end, F-12 worked out great. People admitted that they didn't have all the answers. When someone in ops needs help from someone in dev, they're

willing to say, "Your stupid app is broken. Help me." Dev people are less likely to ask ops for help when they're having design struggles, but we got there. It makes sense to ask since ops will be stuck running whatever they make. Can we scale it? Will it work? Will it be problematic in the long run? If you can get those conversations happening, it works out better.

Start with the end in mind. What are we trying to do? Who will be involved? Avoid that whole toss-it-over-the-fence thing. Some dev teams think they created the next big thing, and ops says, "You expect us to run this thing *how*?"

When does fear of failure cause issues in DevOps?

Not being afraid to come forward about mistakes is valuable. A few months ago, we had a big outage where troubleshooting should have been quick. But, the person who made the error didn't come forward out of fear of the repercussions, so it took two hours to solve instead of ten minutes. F-12 prevented that. **If you mess up, say something so we can get back online. Worry about the cause later.**

DevOps is more about culture than tech. There are tools for everything in IT. But none of them matter without a cultural focus.

What do Agile practitioners disagree about most aggressively?

The IT frameworks and buzzwords. There will be more. **The core of all of them is to create high-performing teams that deliver business value. That's the point of all this. If we didn't deliver something valuable, we wouldn't be doing this in a capitalist society.**

DEVSECOPS

In Silicon Valley, language obeys an unspoken rule called the 'WATA?.' Whereas the trendy ayahuasca ritual gives tech people epiphanies about "product-market fit," WATA? does the exact opposite. It is an eject button for our minds and creativity when we don't want to think about the words we use.

WATA? stands for "What About The Abbreviation/Acronym?" and DevSec-Ops is one of its notorious creations. No idea in Silicon Valley can amount to anything without an abbreviation or acronym, and DevSecOps is no exception.

Development Security Operations in long form, DevSecOps is when you sandwich automated security testing between Dev and Ops. It happens in *Contested Development* Season III, Episode 4, when Dev and Ops accidentally invite Sec to third wheel their date. You know how that happens:

"Sec, why don't you join us for sushi on Friday? [don't actually do that; we're trying to be nice]"

"Sure! I'll bring a date [could hanging with Dev and Ops be worse than playing *WoW* until 3 am?]"

Sec shows up without a date and wants to share all the sushi, meaning Dev and Ops can't order any tempura because Sec is violently gluten-sensitive.

"Why is Sec here?" Dev and Ops think.

"I could have been in Azeroth, but now I get to ask Dev and Ops painfully awkward questions about their relationship!" thinks Sec, oddly gleeful. As

they sit down at the table, Sec asks, "Dev, how exactly does Ops know you're not putting glaring security vulnerabilities in all your code?"

That, in essence, is the role of Sec in DevSecOps. It introduces automated security testing throughout the DevOps process so that speed and iteration don't shoot a company in the foot. The 'why' isn't rocket science, but as anyone who has third wheeled knows, the execution can be.

The Heartbleed bug, for example, was coded into 70 percent of public applications, in part because no one was using security automation to scan for it. We rely heavily on open source code that, despite the authors' best intentions, can have vulnerabilities.

Sec, through blunt questioning, pushes Dev and Ops to reflect carefully on their interactions. AppSec tools, another great WATA, are the barrier that stops Dev from transmitting bad code to Ops. Costly? Yes. But "oops" is a bad PR defense when your company leaks credit cards, social security numbers, and private medical data to hackers.

Some DevOps teams try the old, "someone else's problem" response when asked to take responsibility for security. But good DevOps teams realize that, no matter how snazzy, scalable, and sustainable their products are, it takes *one* breach to ruin a good partnership.

To be clear, we're *not* saying DevOps should take over incident response and penetration testing. We are saying that at a company where hundreds of developers release daily without DevSecOps, *something* will get through and raise all hell in the DevOps relationship. The third wheel is awkward but pushes couples to question aspects of their relationship they otherwise would take for granted.

THE LATEST AND GREATEST ISN'T ALWAYS THE SMARTEST

Kevin Brinnehl

In the early 2000s, the web publishing revolution sparked a mania for content. Demand for digital asset management (DAM) software soared, and Widen Enterprises became one of the first DAM providers to go cloud. Good thing Widen had a team of imaginative engineers, including Kevin Brinnehl, to help make the transition.

Trained as a sysadmin, Kevin taught himself DevOps and left SharePoint, Exchange, and other traditional stacks behind for good. Today, as DevOps and Platforms Services Manager, he ensures that Widen Collective can handle more customers, more content, and more integrations. Here's his take on DevOps:

How do you define DevOps?

Many people try to define DevOps in terms of underlying infrastructure and tools. Going to cloud? Oh, you need DevOps. No, you can do DevOps with whatever you have. It's more of a philosophy than a place. Give me an API to support, and I don't care where it lives.

I think the goal on the ops side is to make sure developers can spend their day coding. We try to make deployment and troubleshooting simple for them.

Why does DevOps matter for Widen?

We wouldn't be where we are without DevOps. It started during Widen's transition from a traditional datacenter to the cloud. As Widen's customer base grew, the lead times to acquire and install new hardware became too long. The cloud and DevOps addressed that issue.

For comparison, at Trek, DevOps mattered for different reasons. There it was about responding quicker to developers. Essentially, the DevOps team was born because developers were sick of filing tickets and waiting three weeks for a response from IT.

How did you get into DevOps?

I joke with fellow DevOps people that I got into it because I get bored easily and I like to be lazy. My journey started in the mid-2000s with PowerShell and branched out over time. The mid-level sysadmin path was boring to me, I knew what I was doing could be automated easily, and there were other skills I wanted to acquire. So, I automated my way out of doing that work.

What is the biggest avoidable mistake you have made in DevOps?

I went into some DevOps projects with the assumptions of a traditional sysadmin and expected things would work the same in the cloud. You can do on-premise IT in the cloud, but you'll waste money. I made some SQL servers

cost four time as much as a better a solution we put in six months later. On a deadline, it's comfortable to do what you've always done.

The other avoidable mistake I've made is automatically upgrading open source DevOps tools. The latest and greatest isn't always the smartest. People spent hours reworking stations after an upgrade. I don't trust upgrades without validating them myself.

What's the most important question you ask your teams regularly?

How can we improve the processes we've constructed? We're never done. There's always new ways, and always new software that improves your build system, decreases deployment teams, and increases uptime. So that's the question: how do we make things better even if the story is completed?

How do you approach monitoring?

We make sure customer services are up and meeting SLAs. It's all based on performance data. For example, we decide an API request will respond in x milliseconds or that web page A's loading time should be no greater than y seconds. We come up with these metrics in consultation with the product team. They'll say they want a new product to have certain metrics. We give it glance and say ok or tell them it's never going to work.

We then go through a series of load tests. As long as we can maintain their metrics through 100 samples, we're good to go. Then we set up scaling factors so metrics can be met. If, for example, a page load time is in the 90th percentile, our scripts automatically spin up a new application node.

How is your job different now from five years ago?

The difference is night and day. I went from doing server admin work with SharePoint to standing up full web stacks using PowerShell and Python. But the biggest difference is the pace of change. If you told me two years ago that Microsoft would open source their .NET framework, I would have laughed you out of the room. They did it a little over two years ago.

Forget what keeps you up at night. What about DevOps helps you sleep at night?

The automation. Our focus right now is on self-healing systems. We construct automation scripts that have health and reliability scripts built in. In other words, if a node in the cluster goes offline, the scripts will kill the node and bring a new one into service. That helps me sleep.

What in the future of DevOps excites you?

The big one is containerization and Google's Kubernetes-based container management engine. It rocks because if a pod begins to take up too many resources, the Kubernetes framework will move it. It can also destroy nodes that fail health checks and bring up new ones. Our goal is to federate Kubernetes clusters between Amazon Web Services and Azure, so in the unlikely event one goes down, that engine would bring up nodes in the other.

Any last thoughts for the DevOps community?

For those resistant to DevOps and cloud, I say get over it. In five to ten years, I don't think traditional sysadmins will exist.

Our community college here in Madison, Wisconsin (Madison College) began to offer a cloud operations administrator degree this semester. Students will come out knowing basic coding and ops. I could hire four sysadmins or one of them. Easy choice.

MONITORING

With no meaningful differences in opinion, techies have waxed poetically about the power of 'big data' for a decade. Reading most of their articles is the hybrid experience of listening to an elementary school orchestra while perusing a Facebook feed. Ooh, Chris is arguing that nutritional yeast tastes better than cheese in noisy, grammar-starved English? I better say the same things but in CAPITAL LETTERS.

That said, we can't escape data in this discussion of DevOps monitoring. The art is to automatically aggregate, analyze, and act upon swaths of application and machine data so you can prevent outages before they happen and write code that is less susceptible to breakdowns.

For most of human history, progress was driven by the question, "How do we make life less dangerous, difficult, and unpredictable?" DevOps monitoring continues that tradition.

The desire to monitor is human, not techy. Our brains developed to read dangerous environments. Interpreting wind, waves, animal tracks, etc., helping us stay alive. Likewise, monitoring Facebook for trolls and unfriending them helps us stay sane. We can't outrun tigers (or digital trolls), but by using the data and tools we create, we prevent ourselves from getting stalked in the first place.

Although the case for monitoring is obvious, it gets cut from IT budgets first – until there's an outage that embarrasses the company. While we'd like to believe that people are patient and understanding, the reality is that they are Pavlovian conditioned to get what they want instantly or feel outraged. We've all had moments where an internet outage feels as stressful as getting chased by a bloodthirsty tiger. And we're quick to bash and stop

paying companies that fail to satiate our on-demand entitlement.

Facebook, for instance, has 2 billion users who log in at all hours. Keeping a system of that scale up requires diligent monitoring. People need to be able to read the heated exchange between vegan Chris and carnivore Dan, who's now taunting Chris with bacon memes.

Monitoring also protects the systems behind hospitals, nuclear reactors, military installations, and services that could cause real disasters, not Tweet storms (we assume that's where outraged people go if Facebook isn't working). More importantly, monitoring reveals how code operates in the real-world.

Here's an analogy: a standup comedian gets instant feedback for anything he says on stage. Either the audience laughs, stays silent, or boos. Laughter, or the lack thereof, show how his jokes (i.e. code) operate outside his brain.

DevOps is subtler. Application performance monitoring tracks memory, storage, and processing power so you know when resources are saturated. Machine data monitoring looks for quirks in application logs and details such as which IP addresses are accessing the app.

That's our jungle, and those data points can predict when we are going to get booed off the stage, mauled by a tiger, or assaulted with vegan propaganda.

PICTURE OF A BEATING HEART

Sarah lahav - SysAid

Historically, DevOps has *not* been gender-balanced. Thankfully, that's changing. At SysAid, led by CEO Sarah Lahav, more than half of the DevOps team is female.

Sarah began her IT career in 1995 at Iscar Metalworking (now owned by Warren Buffet) and was the only woman in the department. In 2004, she became employee #2 at SysAid, helped expand the company's IT service management (ITSM) platform globally, and became CEO in 2013.

SysAid practices DevOps for more than just speed and reliability. Their process is about including customers in the development cycle. Here it is from Sarah:

How did you get into DevOps?

At SysAid, we used to use Waterfall, which resulted in long development cycles. We'd try something, and if it didn't work, we'd go through the process again.

We've always consulted our customers on new ideas and releases – that's one of many reasons why sysadmins trust SysAid. They have an influence on what we do. We used to have focus groups, meetings, and so on. But it became a challenge for us to incorporate their feedback and release quickly. DevOps changed all that. To me, it's a culture where one team, which we

call R&D, absorbs the complete cycle of development from conception to production. R&D interfaces directly with customers for that feedback I mentioned – there's no middleman.

What does DevOps look like at SysAid?

We release every two weeks, and before anything goes live, we've solicited feedback from customers. Basically, on day five or six of the two-week cycle, we push the change to customers willing to test the new feature. We ask, was it helpful or not? Then we reach out to the customers who found it unhelpful.

This live testing is vital because the complexity of our product reflects the environment of the customers. For example, some companies that use our IT asset management tools might have tens of thousands of devices in their system. We test rigorously, but we might not anticipate all the ways a release could affect each internal network. Different security standards, regulations, network configurations, and firewalls could produce unintended effects.

So, DevOps put our QA on steroids. We make small changes on the run, check in with some customers, and make sure it's what they need before going live.

Why does DevOps matter for SysAid? How has it changed the company?

Companies need to roll out changes quickly because Facebook, Twitter, Netflix, etc. have set high expectations for all tech companies – not just consumer platforms. We're supposed to introduce changes all the time yet maintain quality. People want the right experience fast, so to be a leader in our space, we meet those expectations.

Talk to us about security. How does SysAid incorporate it into DevOps?

Customers need to know if they are protected or not. Part of our job is to update SysAid rapidly when a new security threat is discovered. The WannaCry ransomware attack is a good example. Microsoft released patches for WannaCry in March, but when the hackers repropagated WannaCry in May, many companies hadn't patched their Windows systems. It was a disaster.

One of SysAid's key capabilities is patch management, which makes sure every device in a client's system is up-to-date and defended. When we get new patch information, we update as quickly as possible and test – often in a matter of hours – so that both SysAid and its clients are safe. Without DevOps, we couldn't react that quickly and assure our clients they are protected.

How are DevOps and IT service management converging?

We have started to talk about that concept at SysAid. But you have to understand that ITSM has a whole body of processes that are distinct from DevOps.

For example, ITSM might call for getting approval from stakeholders x, y, and z before you make a drastic change to systems they use. That approval could take minutes, hour, weeks, or months to get depending on the organization.

Once you have approval, DevOps is great – you'll get whatever it is done faster and with higher quality. But doing DevOps doesn't absolve IT of their responsibility to consult with the business. If you're going to take a server down, you still need agreement about when, why, and the impact on the

organization. The speed of action change, but DevOps might not change how you make ITSM decisions.

What is the biggest avoidable mistake you have seen in DevOps?

The biggest mistake is that companies underestimate the difficulty of cultural change. They want instant results and won't give it time. The mentality of DevOps redefines the needs of the organization, so if you don't change the mentality,

DevOps won't have a purpose. It just feels like processes.

And there is no "done" with DevOps. You're always changing and improving. It's not something you can finish. At SysAid, we started with longer cycles and shortened them over time. It took practice.

What questions do you ask your DevOps team regularly?

Every Thursday, we talk about incidents. My concern is the customers and anything that affected their service. What happened? Why? How will we prevent it next time? By each meeting, we should have made changes that will prevent last week's incident.

How do you gauge success in DevOps?

You're succeeding at DevOps if you can make significant changes without breaking anything. It's all about minimal time of interference. We want to always be up and running, and DevOps makes that possible. Changing a system is like modifying the genes of a living organism. So much can wrong.

We have an open floorplan, and right in the center of the DevOps space, there's a picture of a beating heart. It's there to remind the DevOps team that they are the heart of the operation. They make sure we have a steady pulse at all times.

CONCLUSION

We never finished the story about the two men fighting in a Las Vegas hotel. As promised, it has a happy ending – and a way forward for Agile.

The original MMA fights of the 1980s tried to answer an unanswerable question: Which martial art was the 'best' for unarmed combat? The performance of individuals became a referendum on fighting systems developed over centuries. They may as well have put priests, rabbis, and imams in the ring to figure out which religion was more 'effective.'

By the late 1990s, MMA fighters had evolved into artless artists. They stopped identifying with Muay Thai and Taekwondo and started adopting whatever techniques and strategies worked. Pragmatism, not servitude to a made-up identity, guided the MMA fighters. Why be a 'Karate guy' only when learning some jujutsu could save you from blacking out beneath an opponent?

The fighters realized there was no 'right' way to fight *if* they wanted to stay conscious and keep their teeth. The pursuit of perfection in hand-to-hand fighting guided mixed martial artists. Towards that impossible goal, they shed the recipe books, identities, and arbitrary rules of the disciplines in which they grew up. Styles evolved from movements into open-source frameworks.

We hope that Agile follows the same path. Right now, we're in that phase of questioning who has the best style. It evades a better question: What is the point of all this?

Businesses go Agile to innovate faster, make more money, and stay in business. Likewise, the companies behind MMA promote fights to make big bucks. But we – the practitioners, coaches, and strategists in Agile – won't stay hungry pursuing productivity and profits as ends in themselves.

Perhaps instead, we can pursue perfection in learning, creation, and quality. That's what great Agile practitioners seem to do. Our craft isn't limited to software development. Wherever a group of people work towards a collective purpose, the Agile mentality can help.

We began this project not knowing the conclusion (hence, not a white paper). We said we wrote this book for three reasons:
To rediscover what Agile is and why it exists.

To show what it's like to practice Agile through conversations with practitioners.

To rediscover common ground in a fractured Agile community.
If you read this book, you now have multiple perspectives on the what, why, and nature of Agile. The common ground is trickier, but based on the interviews, we can identify at least five themes that bond all Agile practices together:

Process emerges from purpose. If you don't know why you're building something, Agile and DevOps processes won't improve your decision-making. Without a why, we're just big-brained monkeys smacking plastic keys so we can get scraps of paper that we trade for food and shelter. If you can find the why, Agile will help you figure out the how.

Being human *isn't* a process. Collaboration had existed for millions of years before Agile came along (arguably, that's why we haven't gone extinct yet). Agile is *not* a magic formula that makes people work together, but rather a mentality. The mentality is that we learn and evolve best togeth-

er. Agile can take us closer to our nature and further from the Dilbertesque corporate cultures that want us to compete, politick, and generally be jackasses to each other.

It looks ugly. Agile does *not* resemble synchronized swimming. The method is chaotic on the inside no matter how routinized it looks from the outside. But the resulting product is graceful. If Agile feels ambiguous, unpredictable, and rambunctious, good. Agile team members aren't sheep. They question, argue, and experiment their way to achievements.

Agile is supposed to be hard. Some people want to be rock stars, A-list actors, or the best at fighting in octagonal cages. But most people with those aspirations *don't* want to put in the work. Agile, like singing, acting, and fighting, takes serious practice. The Manifesto understood the extent to which Agile would upend conventional business culture and therefore meet resistance. Starting a revolution at work is fun. Keeping it alive is hard work.

"Done" is an illusion because quality has an expiration date. What we label "quality" today could be stale in one year. Every "done" in Agile is the starting point for something better. Hell, Agile became a starting point for DevOps, which, in turn, helped Agile scale and sustain itself. Agile resists the tempting notion that you can define and solve a problem completely. Agile "done" is starting over, getting lost again, and finding your bearings somewhere no one else has been.

Baldeep Sadhal, Assistant VP of Customer Experience at AT&T Entertainment Group, made a comment that puts this whole book in context: **"Companies don't die because they can't change. They die because they can't change fast enough."**

Recipe-book Agile, as Jeff McKenna calls it, deceived companies into thinking they *could* change by following the recipes. However, recipes don't neces-

sarily change a culture. The myth that there's one *right* way to "do" Agile and DevOps has become the quicksand where transformations sink. The recipe book absolves companies from changing toxic cultures by measuring Agile in practices, not attitudes and outcomes. To David Hussman's point (in the next chapter), you can't do Agile like dishes. You can only practice agility. That's rough for business leaders who are trained to get 'the' answer and operationalize the hell out of it. But that's the reality.

The Agile lingo will show up in college courses, job titles, certifications, and self-congratulatory corporate mission statements that sound no different from any other. We can live with the lingo as long as we stay on guard against the dogma. Again, the artless art is to be pragmatic, borrow what fits, and discard what doesn't. If there were one right way to practice agility that *couldn't* be improved upon, then by definition it wouldn't be Agile. So where might you, the reader, go from here?

As a start, shamelessly steal the questions that our interviewees regularly ask their teams. Those alone might bring out agility where your teams are suffering from rigidity. Second, continue learning. We recommend reading *The Phoenix Project* and *The DevOps Handbook* for detailed information about Agile techniques, a topic we intentionally skipped. Last, if your development and operations teams are living out scenes from *Contested Development*, our fake reality TV show from chapter IV, do what the couple should have done: Get help. Try DevOps marriage counseling while it's still an option.

GREAT, YOU'RE DEPLOYING THE WRONG THING 'FASTER

David Hussman - DevJam

Many coaches reduce Agile to a rigid set of processes without regard for their purpose. That's why we chose David Hussman, founder of DevJam, for the last word.

No one asks "why?" quite like David. He brings an integrity to Agile that often collides with the alleged 'rules.' In this interview, we question the assumptions and conventions behind modern Agile:

What is Agile? Where does it come from?

Scrum became a popular approach shortly after Alistair Cockburn came up with the term "Agile." People were calling these new methods 'lightweight,' and Alistair didn't like that label so he coined the term Agile. While it was a better name, giving these methodologies a name led to homogenized processes, with Scrum certifications becoming the most popular.

Big A "Agile" can get in the way. It's never been about Agile; it's always been about agility. For me, agility is about finding simple processes that help people collaboratively bond around rapid learning based in evidence.

If people are just doing a process, that's not the point. That's where "I'm Agile" versus "we're practicing agility" becomes an issue. It's the adjective over the noun problem. For example, if you say, "I'm democratic," that's a hell of a statement in the 21st century. On the other hand, "I'm trying to practice democracy" means I have to listen to people I despise because that's at the core of democracy. The same is true of any philosophy, religion, or movement.

I hear many people say first, we'll "do" Agile. Then, we'll "do" DevOps. This seems weird to me because the core of DevOps started when collaborative leaders, some of whom were Extreme Programming (XP) coaches, started reaching out to and collaborating with people in Ops.

More interesting is the use of the verb "to do." In English, you only use that verb in that way when you're talking about something discrete. I'm going to "do" the laundry. I'm going to "do" the dishes. You do it, then it's done. You can proceduralize the shit out of it. I can't imagine Einstein saying, "I'm going to do some physics." It sounds so stupid! The nature of the work is experimental, not procedural. The same is true for many teams building products with Agile practices. Where there is uncertainty, the goal is not to get more done but to eliminate what you do not know.

I often draw continuums while coaching. One of my favorite continuums has things known, things certain, and things procedural on the left. Things that are unknown, the uncertain, and the experimental are on the right. **As a working coach, knowing where people fit on the continuum helps you help them and avoid making them "do Agile." Following the process – any process – does not equal success. Success equals success.**

Where has identity-based Agile led us?

Ward Cunningham, one of the Agile Manifesto authors and inventor of the first wiki, is a mentor of mine. I once asked him, "Aren't you frustrated with what people have done with your ideas?" I was referencing the cargo cult mentality that spawned from the myth of Scrum certifications as opposed to the solid grounding I had learned from being a practitioner of Extreme Programming, which Ward helped Kent Beck develop.

"David," he said, "if it weren't for Scrum, we wouldn't be sitting here." This comment horrified me until I understood his point. Ward was calling out that people need something to latch on to when you ask them to change. If they can say, "Look, I'm doing Agile," they feel grounded. Sadly, it's a bit like hearing someone say "Look, I'm doing Democracy." Or Christianity. Or, if you are into music, it would be the natural tendency of people to latch on to a named set of music like, "We're a grunge rock band."

To cite another great thinker, Warren Buffett once spoke about the causes of the 2008 financial crisis. He explained that first come innovators, then imitators, and then idiots.

I'm not trying to be crabby or mean. I want to help people who want to change, even if they are afraid to change. Most people need a rule set to latch onto to get started. The rub comes when the rules become more important than the impact they produce. I've often used a soccer analogy to help explain this challenge. Soccer is the world's most popular sport because all you need is a ball and two nets, and the rules fit on one piece of paper. But being a great soccer team does not come from knowing the rules. Great soccer teams know the rules, can play the game, and are able to adapt their style to the team they are playing. Adding more rules often gives people the illusion that things will be better, when what often happens is an over-focus on rules instead of impact.

Agile can be a Catch-22 because the more people are asked to change, the more they want rules that will make them feel comfortable.

What do most Agile coaches do wrong?

The number one mistake is that they scorecard how many practices people do and don't look at the impact of the change. Many coaches look at how long it takes to deploy a new feature. While that is good evidence to learn about deployment issues, it fails to measure the impact of the feature. I often tell people, once you start delivering consistently, you need to start asking if you are delivering the right thing.

Good coaching involves finding just *enough* process for a team or a collection of teams working on one product. The best way to figure that out is to look at the team's impact on the market. How many people are using it? How many people pay for it? If it's freemium, how many convert?

When young coaches focus on how fast the team can build, they're not asking, "Are we building the right thing?" With the first question you can say, look, we have zero manual processes! Great, you're deploying the wrong thing faster.

What's your approach to teaching Agile?

When I was a college instructor, I was taught the Dreyfus model of skill acquisition. First, teach them how to do it. Then teach them how to stop worrying about how they're doing it, and start worrying about why they're doing it. Third, watch them improvise and find better ways to do it.

Too many coaches stop at the first level. **They do the practices, have a CI/CD pipeline, and it's exciting, but horrifying in other ways because it's so myopically focused on deployment.** At some DevOps talks, I think to myself, I'm glad you just discovered the automation script I wrote in 2002,

but what else do you have? Or, I am happy you just discovered collaboration, but please be mindful that you did not invent it.

What are the consequences of obsessing over deployment?

The problem I often see in the "DevOps save the day" space is that they're learning how to get better at the process of deployment, but they're not getting better at producing the right product at the right time. Sometimes, there is essential learning that happens outside production or outside the code. In many cases, learning outside and upstream from the development cycle helps teams build less of the wrong thing.

That is not to say that learning in production is not essential. I am floored by the ways that Netflix is learning in production. They have tools that allow them to set up a control group and experiment group, deploy to the experiment group, and compare the results.

When an Agile team has a new idea, and the members are not sure it's right, making intentional choices for learning inside or outside the code is an evolution they need to embrace. If you want to try this, simply take an idea and ask yourself, "What's the simplest experiment we can run to validate this idea?" If you can create an appropriate prototype and gather a significant sample set, you can validate an idea without incurring the cost of going into deployment. That's what I mean by learning outside the code.

How do you formulate experiments outside the code?

Start by learning a bit about DOE, or "design of experiment." Imagine you want to come up with a new chocolate chip cookie recipe. If you add more sugar and change the baking time and temp to 350 degrees for 60 minutes versus 55 minutes, do people like the cookies more or less? DOE asks the person running the experiment to examine which variables they control and which they do not. I meet many people who are conflating experiences

for experiments. Experiments take more discipline, and DOE is a tool to help design experiments that are meaningful.

In the software product and services realm, many things can affect the impact, and designs must take into consideration knowns and unknowns. For example, someone was telling me about customer interviews they were doing that were disproving an idea. When I asked how they determined a sample size, they looked confused. It turned out that they had interviewed ten people when the target audience was in the millions. What they were calling an experiment I would call bad science.

How has your background in professional music influenced your approach to Agile/DevOps?

Most of the influence comes from being a player/producer and not just being a player. Of the many similarities between programming and music, the form of documentation is a strong example. Imagine I show you a piece of sheet music. Can you tell me if it will be a pleasant listening experience? Very few people can look at sheet music and say yeah, it'll be beautiful. And those who can might quickly ask, "Who's performing it and where?"

Writing down music doesn't specify that song or guarantee it will be great. The Beatles wrote down very little. In music, when there's a high degree of collaboration within a small group, you don't need as much documentation. That's the difference between a four-piece rock band and a 300-person orchestra. You don't get a 300-person orchestra to magically gather together and spontaneously jam.

In software, the written music equivalent is the code. Few people, if any, can scan code and tell if it's good or bad. And those who might be able to do that, like Ward, are usually wise enough to say that they can't tell if it will be a

great experience. **You don't know what you don't know until you watch people interact with a working piece of software.**

We're writing this whole book about the Agile, DevOps, and their future. Ultimately, why does this stuff matter?

Sometimes I ask people, "How do you measure how wrong you are?" Too many still stare at me with this glazed look in their eyes. If no one has asked you that, you might be the proverbial frog in a pot. If you assume your ideas and product features are right, you are stuck in the past.

The practices and values in the Agile and DevOps movements are the best tools I found for learning how wrong I am faster. Learning to learn is essential in a world that is ever changing and changing faster every day.

Smaller companies are often agile without trying. Where they are not, or when there are larger challenges, as there are at bigger companies, agility matters for many different reasons. For example, all big box retailer should be terrified that they are not doing as well or better than Amazon. Many retailers are so far behind Amazon that they would be better off killing many of their ideas and instead spending the next six months trying to be one tenth as good as Amazon.

While it sounds challenging to survive a tour of duty at Amazon, the people I know who have worked there talk about never starting a project without an idea of the impact it will have. The ability to more quickly measure the value of an idea is essential to being competitive. Agile and DevOps cultures and practices provide tools for learning fast, as long as the practitioners avoid the pitfall of thinking, "Getting things done faster means we are more successful."

Every organization that's getting worse at keeping up with the rate of change needs to find ways to adapt and learn faster. For me, agility is

fundamental to adapting and learning. Worrying about "Doing Agile" tends to be more about methodology than results, and it is the pitfall that many methodologists have encountered for decades. Next time an Agile purist tells you, "You have to ..." ask them "Or what?" If they have no answer, meaning they can't tell you "why," feel free to ignore them if you can.