BEYOND The Phoenix Project

The Origins and Evolution of DevOps

Gene Kim *and*
John Willis

The Complete Transcript of the Audio Series

BEYOND The Phoenix Project

The Origins and Evolution of DevOps

Gene Kim *and*
John Willis

IT Revolution
Portland, Oregon

25 NW 23rd Pl, Suite 6314
Portland, OR 97210

First Edition
Printed in the United States of America
23 22 21 20 19 18 1 2 3 4 5 6 7 8 9 10

Cover and book design by Devon Smith

eBook ISBN: 978-1942788256
Kindle ISBN: 978-1942788263
Web PDF ISBN: 978-1942788270

Special thanks to Dr. Sidney Dekker, Dr. Steven Spear, and Dr. Richard Cook
for their participation in the 2017 DevOps Enterprise Summit.

Beyond The Phoenix Project

Contents

Module 1

The Phoenix Project

John Willis: Hey Gene, it's been a wild ride. I think the first time we met was actually on a panel.

Gene Kim: That's right! DevOps Days 2010 Mountain View.

John: That's right, and the joke was I was on a panel with you. I think it was Patrick Debois, who we considered the godfather of DevOps, made a joke about my age or something like that. And you made some compliment to me, and it was one of those things where I knew who you were but I didn't *know* who you were. And you said, "Oh, he doesn't look that old!"

Then I got off the stage, and our good friend Damon Edwards says, "Do you know who that was?"

I was like, "I don't know, panel guy number four?"

He goes, "No, that's Gene Kim!"

I'm like, "Gene Kim! Oh, my God!" I was trying to chase you down, and I think that was the first time we met. What was interesting is I think we

agreed to meet at South by Southwest, and we had a really long conversation about this thing we call DevOps. And I think we had a meeting of the minds of what it meant to me, what it turned out it meant to you.

Gene: Yeah, absolutely, and I'll never forget that interaction, because you had told me over drinks that DevOps was important because IT operations has been lost at sea for thirty years, and DevOps is how we're going to find our way back. In that moment I knew that I was talking to a true kindred spirit, and we were genuinely on the same mission of how do you elevate this through the practice—not just for development but also for operations, where I think both of us would say is where we came from.

John: Then we started having discussions about how to further the discussion about this thing called DevOps, and you had basically informed me that you were working on this mission you had about this book that was based on some guy that at the time I didn't really know. One of the other favors you did for me is, I was "Ooh, can I read a copy of the book?" and you said, "John, you should actually read *this* book first." That book was Eliyahu Goldratt's *The Goal*.[1] So I read it, I fell in love with it, and it was just...I think that started our journey of this thing that we've been working together. So, I guess, what inspired you to write *The Phoenix Project*?

Gene: So, as you had mentioned, it is because of Dr. Goldratt's book called *The Goal*. It's a famous book that was written in 1984, and it's a novel about a manufacturing plant manager who has to fix his cost and due date issues in ninety days, otherwise they'll shut the entire plant down. What's amazing is that you see the entire world through the hero's eyes: Alex Rogo, the young plant manager.

What Dr. Goldratt does so effectively in the book is really talk about all the things that go wrong in manufacturing that resonated so well with the

entire manufacturing community, and uses that as a platform to put forth a set of very unconventional, counterintuitive things that he called the Theory of Constraints.

When I read that book, oh my goodness, probably seventeen, eighteen years ago, it was just so powerful to me. And I've never worked in manufacturing. I've certainly never been a manufacturing plant manager. But it was obvious that so many of the lessons being imparted in that book were relevant to the work that we did all the time in technology. I think the other thing that resonated was the notion that storytelling would be such an amazing vehicle, to be able to tell the entire world that this is the problem that we're trying to solve through things like DevOps.

I think the last thing I'll mention is that the impact of *The Goal* is difficult to overstate. It's been integrated into almost every mainstream MBA curriculum, into almost every mainstream Ops management course in terms of operations management. So, I think it just shows how powerful storytelling is as a vehicle to get people on board.

John: Yeah Gene, the whole storytelling narrative of *The Goal* and then *The Phoenix Project*, I think it's really interesting. Because one of the interesting themes that always comes out when people talk to me about *The Phoenix Project* is, "How did Gene sneak into my building?" Then I'll ask them, "Did you read *The Goal* first?"

As some people [say], "I haven't read *The Goal* but I have read *The Phoenix Project*." Even when I read *The Goal* the first time. You said, "John, read *The Goal* first," which was a gift that you gave me. Because I think if I would have read *The Phoenix Project* first, I wouldn't have had the sense of the narrative moving forward. But in both books, even when I read *The Goal*, I got the sense of, "I know that character. I know that character."

I love when people tell me, "It seems like Gene snuck into my building when he wrote *The Phoenix Project*."

Gene: Yeah, it's so interesting to study Goldratt because he did talk about the years after *The Goal* was published, and he said he would get these letters in the mail from people saying, "You must have been hiding in my manufacturing plant. I know these characters, I know these disasters. That person that is being described is me!"

So, I think that was just a great testament to how great of a vehicle *The Goal* was for being able to say, "Hey, I understand the problems that are happening." So, I think a design objective of *The Phoenix Project*, of the co-authors—so that was Kevin Behr, George Spafford, and myself—we obviously want[ed] to create the same reaction in our readers, saying, "Holy cow, what's happening to Parts Unlimited," the fictitious company being depicted in *The Phoenix Project*, "is what's happening to us."

It doesn't even matter whether we're a developer or in operations or security, whether we're in technology or within "the business." The story being described is what we feel every day. I think an additional design objective was to say there's this downward spiral that is happening in the organization, and it's actually getting worse over time. So, that was something we wanted to bring to life in *The Phoenix Project*.

John: The other thing, I think...it's all about timing. The first DevOps Days in Mountain View where we met in 2010, there was this explosion of people trying to figure out a better way to do things. Meanwhile, you had been working on this project for a number of years that just fell in at a perfect time to describe a narrative of what people were experiencing in this thing called DevOps that was just starting to explode.

Gene: Yeah, I'll tell you what I remember from DevOps Days 2010 Mountain View was the two movies that Patrick Debois put out to open up the conference. One was that Charlie Chaplain assembly line scene [*Modern Times*], and then the other one was the *I Love Lucy* scene on the assembly line of the chocolate factory ["Job Switching"]. And what a great way to signal that we're all...we have these Lean principles in mind. In that first ten minutes of the conference, I had the immediate feeling of, *Holy cow, I'm surrounded by people that probably believe the same things I do*.

John: So, like I said earlier, you had introduced me to *The Goal*, and I started reading...in fact, even before I actually read the early copy of *The Phoenix Project*, I was already off. I was reading all of his [Goldratt's] works. And they all seemed to have this narrative, like you said; there was this hero and this conflict and there was something that they had to get done. Then the storyline usually has a mentor. So, there's always the hero, and the thing I guess I would ask you, what's your sort of favorite scene in *The Goal*?

Gene: This is something that Goldratt does in *The Goal* that is just, for me, was very meaningful. There was this scene where our hero, Alex, is running to a conference, and he's running through the airport, and he's in the airport lounge, the airline lounge, and he runs into his old physics professor named Jonah. So, Jonah is really the incarnation of Dr. Goldratt.

They catch up on old times, and Goldratt [Jonah] asks, "So, what are you doing?"

Alex says, "I'm a manufacturing plant manager," and he's going to give a talk at a conference.

Goldratt [Jonah] asks, "What are you presenting on?"

Alex says, "I'm presenting about how robots have improved my efficiencies."

Jonah [Goldratt] looks immediately skeptical, and he says, "Oh, really? How have robots improved your efficiencies?"

He says, "We've increased efficiencies by thirty-six percent."

And Jonah's first question was, "Are you selling thrity-six percent more products?"

Of course, Alex's response is, "No, no, of course not. We're manufacturing, not sales. That's not our job."

Then Jonah [Goldratt] asks the second question, which is, "Did you lay off thirty-six percent of your people?"

To which Alex responds, "Oh no, of course not. We're a union shop; we can't do that."

John: I think it was funny because as I was reading...because that is one of my favorite scenes too, it was almost like Alex was telling him, "Oh, you don't understand manufacturing," which becomes the kind of joke.

Gene: Exactly right. And I think the point is, with the third question clearly Jonah understands manufacturing. In fact, he says, "If you haven't increased your sales by thirty-six percent and you haven't decreased your cost by thirty-six percent, then you haven't really improved your efficiencies at all. In fact, I bet your inventory levels are going through the roof, aren't they?"

And in the book, Alex says, "It felt like I was in an elevator and the elevator cable just snapped." Alex knew that he was caught in some sort of logical error, but didn't know exactly how. So, I think this really shows that Goldratt had an internal mental model, and what he was doing was looking at the inputs and outputs and work in process, and the only way you can increase efficiency technically is to increase throughput or sales, decrease costs. If you haven't done that, what you're actually doing is creating more work in process, which leads to all sorts of horrendous things like worse due-date performance, increased lead times, and so forth.

So, without a doubt that is my favorite scene in *The Goal*.

John: I think it takes a physicist turned management consultant to actually go ahead and explain the simplicity of what seems like an abstract, complex problem. I think the reason that's my favorite scene is in every book you read of the Goldratt series—and other people have written books around this narrative, and of course *The Phoenix Project*—there is always this mentor that seems like the person who's out of place but then becomes the Socrates or the dialogue. He doesn't give you the answer. You shall figure out the answer yourself.

Gene: It's funny, that is actually one of the tropes that we used in *The Phoenix Project*. That notion of a mentor is very widely used. It's Yoda in *Star Wars*. It's Mr. Miyagi in *The Karate Kid*. It's the person who seems to know everything, and their mission is to guide our hapless hero to answer the puzzles so that they can achieve their question.

John: So, like you said, Gene, one of the things about *The Goal* was you didn't really have to understand manufacturing to get a sense of the things, the issues, the flow, the Theory of Constraints, those things that happened in there. One of the things over the years is we've had this journey together from reading early copies of *The Phoenix Project* following this journey with you. One of the things I realized is the brilliance of what you produced was to be able to transpose those ideas into a modern day software world.

I thought about what Mary Poppendieck did with her famous book *Lean Software Development*, and I also thought that was a brilliant thing that she was able to take Lean manufacturing and kind of transpose that to software development in such a way to connect those two. Now I look at what you did from a novel that I guess some, an engineer, would look at as automated robots and enclaves and you were able to turn that into...make that leap, very

much like Mary Poppendieck, in modern day software company with software issues instead of an enclave maybe, system programmer and a late project.

So, how were you able to do that?

Gene: You bring up Mary Poppendieck's great book *Lean Software Development*[2] and I love the—I think the interview that you and Damon [Edwards] did with her where she said she was at 3M and she was actually writing the systems that supported plant manufacturing. She said, "I learned so much, and we should be applying these same Lean principals to the software development process."[3] I think that was very much our similar goal. Take all the things that we all took for granted as absolutely true in manufacturing and apply them not just for development and keyway but also for the IT operations and service delivery. In fact, you could say that one of our design goals was to create an isomorphic mapping between the Lean principals as applied to manufacturing and the Lean principals as they applied to the entire technology value stream, which I think that we actually made even more concrete in the book that you and I co-authored together in *The DevOps Handbook*, along with Jez Humble and Patrick Debois.[4]

You put your finger on, I think, probably the biggest insecurity going into *The Phoenix Project*, which was the fact that we didn't come from manufacturing. For us to really learn that skill, there are three places that were very meaningful to me. One was a course I took—actually, it was two courses, three courses at Washington State University in 2008. That was three engineering management courses taught by Dr. James Holt that were all about the Theory of Constraints, the applications of Theory of Constraints, and was actually based on the training that they used to call the implementation certification that was issued by the Theory of Constraints, TOCICO [Theory of Constraints International Certificate Organization]. So,

George Spafford and I went through that together. Then later in 2011, George Spafford, Kevin Behr, and I, we all took the Toyota kata training from Mike Rother, and that was in 2011 at the University of Michigan. So, that was a phenomenal course. That was only four days, but it was two days of lectures as well as two days in the field on the plant floor.

Those courses really gave us a lot more of the theoretical underpinning of some of these principals, as well as some of the more tactile learning about what does a manufacturing plant look like and smell like, and hopefully those were reported accurately in *The Phoenix Project*.

John: You know, it was interesting too, we talked about the work that we did together, *The DevOps Handbook*. After I had finished reading *The Goal*, and even to a certain extent some of this stuff in *The Phoenix Project*, I wanted to learn a lot more about Theory of Constraints. I think I came to you a couple of times, and you were "Yeah, I've got a lot of this research, but it's here. It's here." It wasn't easy to find a lot of deep explanation in that, but then during the project in *The DevOps Handbook* we were able to share a lot of that stuff. I was able to see that, and as we talked about case studies and things in *The DevOps Handbook* I think it became a lot clearer, some of the information. Again, I've been drafting off of your knowledge for many years.

Gene: In fact, I think one of the criticisms of the Theory of Constraints body of knowledge, and justifiably so, is it is a little bit hard to access. Rather, it is somewhat inaccessible. In fact, what I learned in the Washington State University graduate courses is that there's still professors working on trying to make it understandable and applicable. There's certain things only Goldratt seem to be able to do.

So, I think that work still continues. Kind of an ah-ha moment for me is that it's one thing to put out a great theory, but if you don't have a vibrant

community out there who can actually deploy that into industry, then it really becomes an academic, more intellectual work.

John: That's what I was saying earlier. I think there are bodies of work that are really important, and then the important part becomes the people who can actually start explaining it en masse. I think that's a little bit of another narrative of how we do things even in DevOps, our community of DevOps, and even about this body of work that we're working on now.

Gene: Absolutely. I think what I get very excited about is here's a way that we can take all of these over a century of disparate bodies of knowledge and show how they're all converging to help make DevOps possible, which I think is, in my mind, the first time that these principals are being deployed at an industrial scale.

John: Well, and then I think too, just to summarize this, *The Goa*l, I think why we we're both kindred spirits is the real goal is—excuse the pun—but is that we both have been working in this industry a long time, and I think what DevOps brought to light for us is the ability to finally help organizations try to become high-performing organizations. It wasn't just one set of principals or disciplines or frameworks. It took all of this work to put together and be able to explain it in such a way.

I think the recreation of that scene in *The Phoenix Project* is my favorite. That one like what we had with Alex and Jonah, the mentor, we had a similar scene with Bill and Dr. Erik Reed where basically, again, there's this whole discussion, and he [Bill] says, "You don't know what work is." I think it was Ben Rockwood who wrote an article that described this idea of the four types of work.[5] Can you describe these four types of work and why it's important and why it fits into the DevOp so well?

Gene: That's probably one of my favorite parlor tricks that we used in *The Phoenix Project*, and it was really deployed when our hero, Bill Palmer, says, "Who is this buffoon? What can we learn from manufacturing? [In] manufacturing they work with [their] hands but we in technology work with our heads." I think it was really an attempt to sort of verbalize all the objections that we would hear about what possible lessons can be learned from manufacturing. So, the Jonah character, Erik Reed, says, "Well, clearly we're not ready to have this conversation, because you don't even understand what the four types of work are." That was really meant as a puzzle that the protagonist has to go and fix.

It's worth pointing out that it is true. There's actually a whole bunch of problems that you see in manufacturing that you can't see in technology work, where work and process is physical. It's stacked clear up to the ceiling. You have to walk around it when it clutters up the plant floor. So, the point of this exercise was to show that there are four different types of work. It is a very special category that has all the negative properties that work in process does in manufacturing.

So, the four types of work, the first one was the business project. These are the business initiatives. They're on the annual PowerPoint slides that are shown by executives, and those are almost always backed up by a development project. They're typically tracked in a project management office. They're being tracked as a point of investment capital. So, those are pretty obvious, very easy to count.

Then the second type of work are the internal IT projects. These are often infrastructure projects, maybe operational improvement projects like deployment automation, like production telemetry. The interesting property of these is that they're not often centrally tracked. They might be tracked at the divisional level or within a certain silo. They're typically tracked by the budget owner, but they don't get rolled up anywhere. So, for anyone in the

technology value stream, a lot of our work is consumed by this second type of work, but it is not as visible.

The third type of work that we posited was operational changes. These are really the result of the first two types of projects, so both business projects and operations projects to get itemized into changes that have to be done by a whole bunch of different work silos, whether it's DBAs [database administrator] or network engineers. So, this type of work is increasingly more and more distant from the business objective, and they're often tracked in totally different systems than the first type of work. So, it's often in ticketing systems like Jira, Service Now, or Remedy as opposed to the capital projects.

But the last type of work was the most fun to play with, because this is unplanned work. This is all the work that's generated from failures, outages, audit, and security compliance findings. The point of this fourth type of work is that left unchecked, unplanned work can consume the work capacity of the entire organization. So, it really isn't tracked anywhere except for problem tickets. I think in some schools this is called failure demand as opposed to project demand.

The scene that I really love in *The Phoenix Project* is when they're tracking all the work cards on a kanban board, the unplanned work caused by the failed Phoenix Project launch ballooned and no other work could get done. So, these work cards were piled to the ceiling just like work in process was piled to the ceiling on the plant floor. That was really the whole point of having Bill, the protagonist, have to find the four types of work.

John: I think the other interesting thing, you talked about Mary Poppendieck and how did she make that translation. She was at 3M. We have another friend, Scott Prugh at CSG, and I think one of his early...he speaks at the DevOps Enterprise Summit, and I remember one of the times

he said that he realized—and I think actually at the reading of *The Phoenix Project* and understanding that really there is a mapping between manufacturing and software delivery or service delivery, I think he said in one of his early presentations that he went back to the manufacturing floor—which we're all kind of Theory of Constraints, Lean, delivery—and he spent some time with them to come back and take that.

I think that going back to being able to map the goal manufacturing to a software service delivery is, I think, too many people look at the works in manufacturing and think that's just manufacturing.

I think that becomes the common theme that me and you are so interested in. If we take the abstraction to a value stream, then we're able to look at Lean, we're able to look at Goldratt's work, we're able to look at a lot of different bodies of knowledge that don't get muddled in just rote communication in those kind of silos. If we move it up and we keep talking about the delivery of a value stream in an organization, we're able to include a lot of different bodies of work, which actually is one of the goals of this project.

Gene: Absolutely, and by the way, you had mentioned Scott Prugh at CSG. It was such a treat to be able to go visit one of two of the largest printing plants in the United States. He took me to one of them outside of Omaha, and he showed me exactly where he had one of his many epiphanies, which is at the starting line of the print jobs. Basically every print job corresponded to a cart, and they would not let a cart go onto the production floor until it had every one of the things required of its dependencies already fulfilled. In other words, the worst type of thing that could happen if you're printing hundreds of millions of bills in monthly statements every month is to let a job proceed onto the plant floor and then realize in the middle that you're missing a certain type of ink.

The parallels that he drew was that notion of flow control and job release was something that was critical in software delivery, especially in operations. How many times has a deployment started only to realize halfway through, maybe two days into the release, that we're missing something. We're missing a database scheme of change, or we never notified the firewall team that we needed a rule change? I think that was a very concrete epiphany that Scott Prugh had, and it was such a treat to be able to see that in person.

John: I think there's a little foreshadowing for some of the stuff we're going to cover in the future about quality, and build[ing] quality in early, and some of the bodies of work that are going to teach us early, early examples, even going back to Japan, about that type of concept of being able to not let the bad ink go down the road in CSG's example.

One of the things that came out of *The Phoenix Project* is this idea we've called the Three Ways, Three Ways of DevOps. What's interesting is it's become a way for us to explain this, in a very simplistic way, of how we add value, how we add service and value stream, and how we think about it. I really do now think it's become...the Three Ways have become this common language of DevOps. Can you explain the Three Ways?

Gene: Yeah, I love that word that you used. It's very simple. It's simple to explain, and I think it's because the Three Ways are simply principles. The goal was to be able to say here are three principles from which [we] can derive all the observable practices. We can, from those, derive all the things that we see in high-performance technology organizations, whether it's a Google, Amazon, Facebook, or a Nordstrom, CSG, or Capital One.

Just a little trivia fact about the Three Ways is the Three Ways actually changed a little bit. Sometime in 2014, Tim Hunter, who's at Apple, basically rewrote them to make them alliterative. His version of the Three Ways says that the first way is all about flow. The second way is all about feedback. The

third way is all about a continual culture of experimentation and learning. I loved it so much that we actually went back and changed the Three Ways in *The Phoenix Project* to reflect this more pithier, alliterative version.

John: I think there's an interesting story there in that when we first started talking about *The DevOps Handbook*, *The Phoenix Project* wasn't even complete. I think we've learned a lot over the years what the...We use the phrase continuous delivery, but even in the early days if we, we first started talking about this in 2010, 2011, what we thought about the value stream then and how we've gotten to learn from other people and see it go through. I think that now you're talking about the new definition, if you will. It's a nice journey that we wind up to where we understand now what these ways really mean as we continue to learn and explain. Describe for us the first way if you will.

Gene: The first way is all about flow as we go left to right in the value stream. Typically, in the technology value stream, that begins in development, goes through keyway, goes through operations and deployment, so that service is actually running in production so that customers are actually getting value. The goal is to—just like in manufacturing—maximize flow. To do that, we involve practices like reducing batch sizes, integrating quality into our work, and incidentally, also security. By doing that, that's how we end up with these amazing outcomes that we want. We have world-class reliability, availability, and security.

I love all the technical practices that you can map to the first way: the notion that we shouldn't pass defects on to a downstream work center; the fact that often when we do work, our customer really isn't the end user customer, but we should be optimizing it for whoever we're giving our work to next, the internal customer as Lean defines it; the notion that we want to be continuously building, continuously integrating, continuously testing,

and maybe even continuously deploying; the whole notion of reducing work in process. We know that the more work in process we have, the worse the due date performance, the worse the lead times, and the worse the quality. All of those things we would fit underneath the first way. Those are some example practices.

John: If we're successful in our kindred quest to help create high-performance organizations, we try to help companies understand a value stream. If we help them then look at other areas of the organization, systems thinking—as Goldratt would say, global optimization—then we're successful in our conversation. I think there's a great story of one of...an organization that we've worked with, Nordstrom...I think one of my favorite examples of a first way example, and me coming from a mainframe background...I'll let you tell the story, but this story of how they would look at a value stream in a certain way over time every year. It was a mainframe Cobol app, and every year they'd look at it, and they weren't able to look at it in a holistic global way—in other words, trying to add in Lean principles and things like that. One year, a good friend of ours, and I'll let you introduce her, took that idea and just added the Lean principles.

Gene: I love this. This came from our mutual friend Courtney Kissler, who was at that time the VP of Technology for all the non-discount properties at Nordstrom, and this great story that she tells us in *The DevOps Handbook* is how they had this huge problem around the cosmetics business office. Everybody knew for a decade it was because it was that mainframe application. Every year they committed themselves to retiring and getting rid of that mainframe app. They did a value stream workshop, and they actually invited everyone that was associated with every aspect of the creation and service delivery around that mainframe app. They found the problem wasn't anything to do with the application on the mainframe at all.

It was the fact that every time some operation needed to be done, the store floor manager would have to go to the back office and use a PC, and it was very inconvenient for them to do so. They would do it once in every great while, and they would batch up the operations.

The second thing they found was that on that application, they were asking for a piece of information that the store manager didn't have, like the employee number of the specific employee that they had to get into the system. So, they did this experiment. They actually mocked up an app on an iPad, and the lead times went down from fourteen days to get the employee discount codes created, to it being done almost in real time. Here's an example where it had nothing to do with what they had thought it was. Instead, it had to do with information availability, and the common measure was so simple compared to what they thought they had to do.

John: There again is a great example of looking at a problem from an IT-centric perspective. This was a mainframe Cobol, which almost every Fortune 5,000 company has, and they were looking at these horrible lead times and just all these kind of restart of high MTTR [mean time to repair] and the brilliance was to just look over the fence and look at something...Mike Rother's *Learning to See*,[6] which was the description of a principle used by Lean, which was the value stream mapping. By just making that bridge from IT-centric, mainframe Cobol, "Leave it alone. There's nothing you can do with it," to "Oh, I can take a manufacturing principle and find out that it's an incredibly simple problem."

Gene: We changed a form and all the problems go away.

John: That's correct. In fact, from what I heard, in the middle of the value stream mapping they went ahead and wrote a JavaScript program that actually automated the process that was manual.

Gene: That's awesome. Maybe just as a little side note, one of the edits we made to *The Phoenix Project* was because of a comment that Jez Humble made, saying that that exact process where the team does a value stream map of how things are deployed in the production was exactly the genesis of continuous delivery and is described by the seminal paper that Jez Humble and Dan North wrote back in the 2000s.[7]

John: And Chris Read.

Gene: And Chris Read, that's exactly right.

John: Chris Read, Dan North, and Jez Humble. In fact, what was so interesting about that, if you look at the Wikipedia page for continuous delivery, it has that picture that those three gentlemen presented, I think it was Agile 2006, and to me, that's always this brilliance of being able to...I show that picture now in slides, which is the value stream flow with the gates. You have your check-in, your testing, and then you got a red block, which means it was a gate. It gets kicked back, and then you get green, green, green, green, another red block, and gets kicked back. When I present that slide today, I still get oohs and aahs. To then explain to them, "By the way, Chris Read, Dan North, and Jez Humble presented this picture in 2006."

Gene: Right. That's how Jez Humble got added to *The Phoenix Project*.

John: In fact, there's probably no better single way to codify the first way than that image of the gating project of going left to right.

Gene: In fact, as you had mentioned in numerous conversations, the first way I think really does encapsulate so much of what has now been codified in continuous delivery, continuous deployment. That is definitely a first way pattern.

John: It's all about the value stream. Gene, now can you describe the second way for us?

Gene: The first way is all about that left-to-right flow of work. The second way is all about the reciprocal flow of feedback. The notion is that whenever something goes wrong—or for that matter, whenever something goes right—we want to radiate that feedback to everybody in the value stream, and we want to amplify that feedback and make sure that we can create feedback as soon as possible, especially at the earliest parts of the value stream. In our work, that would be developers. How can we create the fastest feedback possible so that they can learn from mistakes, learn from successes, so we can see what worked, what didn't work?

I think for most of us, where we want to concentrate feedback first is when things go wrong. The idea there is that we want to create the right sorts of feedback so that we can ideally prevent bad things from happening again, and if we can't prevent it, at least enable quicker detection and recovery.

One of my mentors, Dr. Steven Spear, he has this notion that he created from the Toyota production system. He said one of the key attributes of high performers is that they see problems as they occur. They can swarm their problem until there's an effective measure in place, and the goal is to take those opportunities, not just to fix a problem, but to enable learning.

I think the technical practices that we can derive from the second way include the notion of assertion statements in code. We have a certain assumption of how things operate in reality, so we want to test them. We want to continuously test, continuously build, continuously integrate. I love the notion of everything radiating telemetry. That's the continuous deployment pipeline as well as making sure that everything in production is actually radiating as much information as possible so we can actually see [if]

things [are] behaving and operating as designed. By doing that, we can actually create genuine shared goals that span Dev, QA, operations, and information security.

By the way, John, one of my favorite parts of working on *The DevOps Handbook* with you was showing how telemetry isn't just for operations people, that telemetry is for everybody.

John: I think what's interesting, just starting to study this body of work, starting with *The Goal*, and everything I've studied since...It was interesting, almost like the Bill/Eric Reed thing, you don't really understand monitoring. Because I started in this business in 1980, and most of what I've done for many years, up until I've studied things like *The Goal* and value stream, was monitoring for monitoring's sake, event for event's sake. And it wasn't until you start thinking of the value stream and global optimization...

To me now, the second way is a really good way to describe why we do this, because we're amplifying feedback loops. We're not just sending out alerts. When I think about all the monitoring tools I've used...to think of it now in terms of telemetry. I think it was your section where you created in *The DevOps Handbook*, where you call it telemetry. And I know we gathered that from lots of conversations, but the light bulb went off for me about what I had been doing for the prior twenty years and why I really should be doing this, which I think is the second way to me.

Gene: And when we were writing that section, what became...what was startling to me was the fact that you had pointed out something that, holy cow, we've all been doing: logging, fault management, events management, but it was all typically in our silo. The Ops [operations] people had all the fault management tools, and developers had the logging frameworks, but they were rarely joined together in a way that we could use to meaningfully

solve problems together. It was actually that section...I remember working on that with you, where it was a genuine eye-opener.

John, it's funny in hindsight...I think the second way is interesting for a couple reasons. One is certainly during problem resolution. If you have an hour to do service impairment, it certainly helps to see what you're doing when you're fixing things. That seems pretty obvious in hindsight. But the second part is that telemetry is actually necessary to even conduct experiments. So, when we talk about continual learning, continual experimentation, the second way is absolutely a prerequisite.

John: I think that works really well into the third way, which is that you need the second way to get the telemetry because the third way allows you to create the completion of an experiment or hypothesis that allows you to learn. Can you go ahead and explain the third way?

Gene: So that first way is flow of work. The second way is about that feedback and telemetry, and the third way...I don't know about you, John, but this is where I've had the most amount of learnings in this journey together with you. So, the third way really is this culture that fosters risk taking, so we can learn from successes and failures. It says that repetition is a prerequisite to mastery. But then it also says it's about a culture that creates a continual experimentation and learning culture. I think there would be many people who study high performance who say, ultimately, for any high-performing organization, improvement is a byproduct of organizational learning.

There's a school of thought that says how high performers win in the marketplace is because they out-learn the competition. So the patterns that emerge from this include that notion of risk taking, safety culture, blamelessness, but that is what sets the stage for us to be able to do experiments.

I love this quote, again, from Dr. Steven Spear. He says, "The goal of leadership is not to command, control, berate, intimidate, and evaluate workers through some set of contrived metrics. Instead, the job of leaders is to help organizations become better at self-diagnosis, self-improvement, and to make sure that local discoveries can be translated and converted to global improvements."[†]

So, I think about all those things. And we look at what's happened at Etsy as they went from a low-performing organization to a world-class engineering culture. So much of that is all made possible through a learning culture.

John: I too think the third way is really the most interesting. Because we talk about things never being done and learning never being done. And it goes back to our journey, to keep understanding the learnings of these things like value streams. And I think it's Andrew Shafer, a good friend of ours, says, "You're either a learning organization or you're losing to somebody who is."[9] And I think this is the area where, when you open yourself up to, in Japanese culture and in Lean we call it Kaizen. But it is this idea that we never end. So, it goes back to...if you think about the first way, going left to right; if you think about the second way, right-to-left telemetry; the third way really codifies the way that high-performing organizations think. They do things in an experiment mode. They have an idea, they test it, they check their results, and then by the results they were able to learn how to make the next step, which is a constant cycle of learning.

This is a place where, in this body of work, we're going to spend a lot of time, but we'll never be done with this section.

Gene: And I think—as if the first way and the second way weren't crazy enough—I think that the truly crazy stuff comes out of the third way. Right?

The notion that Netflix was so passionate about reducing the points of failure and decision that they can only survive failures by failing all the time. This is when they've told the world about doing Chaos Monkey, where they randomly kill entire computer services in production, which is how they survived the first massive cloud failure in 2011. It's the notion that we care so much about availability that we'll randomly crash services in production because we care about availability.

They're these counterintuitive notions of things, breaking things to make things better. But I think one of the things that's always exciting to me about this project, in general, is that we're going to cover all these bodies of work that really fit into the third way. Looking at other disciplines, tying that back into how, just like our early airline story, how you can look at not only Lean but other things we're going to cover in this to just add to this continuous way to learn and absorb and just evolve.

And I do love the fact that, without a doubt, and some of the projects that we've been working on and that we're going to elevate in this project, is the notion of a dynamic learning organization. There's so many rich practices and principles that we can pull in from there.

John: So, one of the things that I love that I learned as part of this project, that I learned working with you, is the detail of the craft of how you created *The Phoenix Project*. Like I said, I earlier read *The Goal* and then I read *The Phoenix Project*, but as we started working on this project, I learned a lot more of the depth of how you created...Often you use the isomorphic mapping. Can you go into the detail of that?

Gene: Yeah, I think one of the things that we were laughing about was your reaction when you discovered to what extent we copied the structures of *The Goal*. I think one of the obvious examples of that is the fact that there's 170 pages in *The Goal* of just describing the problem. As I called, I said you have

to get to page 171 before they offer...Yeah, Dr. Goldratt, it's the first time that he actually discusses anything dealing with the potential solution. So, as we were writing *The Phoenix Project*, we too decided to write about the problem for 170 pages before there was one inkling of what the solution might look like. A lot of that we learned through taking these courses from Washington State University from Dr. James Holt.

And we even knew before that...that in *The Goal* there were five breakthroughs that the team made. Five times that the constraint moved where the team had to figure out where'd the constraint move to, how'd they elevate the constraints, subordinate everything to constraint. So, in *The Phoenix Project*, there's exactly five times that the breakthrough...five problems where some breakthrough had to be made. So it's really difficult to overstate just how much we copied the structure of *The Goal*.

John: One of the interesting things for me always is, I'll ask somebody, "Did you read *The Goal* before you read *The Phoenix Project*, or did you read *The Phoenix Project* before you read *The Goal*?" And what's funny, some people —not all—will say, "It sounds like this Gene Kim guy copied *The Goal.*" And then I've gotten even...*The Goal* scene was "They ripped off Gene Kim." And then I go into this explanation of the purposefulness of how you created this book and the detail.

Gene: It's funny, you can call *The Phoenix Project* really an homage to this incredible work that Dr. Goldratt created, which many consider *The Goal* to be one of the reasons that allowed Lean manufacturing to cross the chasm. It was the first time that there was really widespread understanding of what these principles were about. But there were a couple of times in the writing of *The Phoenix Project* where there were scenes that seemed a little bit too familiar, and then we'd reread *The Goal*, and it's like "Oh!" Because we

actually stole the scene from *The Goal*. So we actually had to take those scenes out. There's a difference between homage and copying.

John: So, as we've gone through your inspiration from *The Goal* to *The Phoenix Project*, one of the things I've really liked that we've talked about is how you see it as this hero's journey and any other inspirations you have.

Gene: Yeah, in fact, the hero's journey...What's interesting about the fact that there's 170 pages of problems, it really does map into a very specific shape of a story. There's a great Kurt Vonnegut talk[10] about that fact that there's only certain finite numbers of story shapes, and probably the most famous and obvious one is the hero's journey as described by Joseph Campbell. And, so, the major segments of a hero's journey [are] you have a hero who faces a problem, who faces despair, and then goes to the mountaintop to seek enlightenment and is aided by a mentor. And then through the help of a mentor, and maybe a fellow cast of characters, he then achieves the quest—slays the dragon, conquers the kingdom, or whatever. But what was really neat about the way that *The Goal* was constructed was it really does match that hero's journey. That the purpose of the 170 pages is really to describe in extreme, vivid details the problem, the forces, that every manufacturing organizations faced. And our hope was that, in *The Phoenix Project*, we could describe in equal clarity every sort of problem that every functional silo in the technology value stream also faced.

John: Any time we can add Vonnegut and Joseph Campbell into a work, we've really done some good work here.

Gene: Fantastic.

John: So, I guess what I would ask is, any final inspirations that you've gathered from all this?

Gene: Yeah, there's one that I think is worthy of highlighting. That's a specific model that we incorporated into *The Phoenix Project* that I think everyone should know about. It's called the COSO Cube. It actually came from, of all places, the audit community, but it's also used a lot by governance people because it matched the language that boards of directors typically organize themselves around. The COSO Cube says there's really four things that every organization needs to care about to survive and win in the marketplace. The first one is, can you produce accurate financial reporting? In other words, if you're putting out a quarterly financial statement, whenever you're putting out the reports, can you make sure that all the account balances and values are accurate? Is there any money missing? So, that's certainly one thing that any organization has to be able to do.

The second one is, are we compliant with all laws and regulations? In other words, whether it's US export laws or things like payment card cardholder data, whether it's compliant with labor laws, we have to show that our organization isn't breaking any rules and regulations or contractual obligations.

The third one is about operations. Can the organization perform all of the critical business processes that it needs to in order to sustain itself in the marketplace, and are they working effectively and efficiently?

There's a fourth one that says, does the company have an adequate strategy to be able to survive as an organization?

What I love about *The Phoenix Project* is that we actually created problems that show that all four facets of the COSO Cube were in jeopardy. They had SOX compliance issues showing that they couldn't adequately assert that the financial reporting was accurate and timely. We showed that they were in violation and breech of the payment cardholder, the PCI regulations, and contractual obligations. And, third, we showed that the

company couldn't effectively and efficiently perform their most critical business processes. So there was a very fun way to make sure that we could convince everyone that the problems that were happening at Parts Unlimited, they weren't IT issues, they weren't technology issues, but they really rose to the level of being some of the most important business problems that the organization could have.

John: And this is so timely. We think about today, with all of the breeches and real publicly loud breeches in security. And now, what we've learned after the last couple of years of the integration of DevOps and security as a discussion that you've been a driving force of...it's interesting that you were able to implant that stuff from, again, when we look at the manufacturing silo and be able to cross over and even add the stuff that was...is so timely right now of thinking about a value stream in a global...a system's view to the point where security, as part of the delivery, is so important.

Gene: Two of the biggest technology stories in 2017 are probably the Equifax security breach and the British Airways technology failure. So, with Equifax, they lost hundreds of millions of customer data records, and let's talk about them not in the concrete but in the abstract. We have friends there, after all. But the point is, they viewed this as a security failure or a technology organizational failure as opposed to a business failure. In my mind, when you lose control of data that you have custodianship after, it is of existential risk when you lose control of that data.

I think, in my mind, another great example of this was in June 2017. There was a data center problem where British Airways couldn't conduct its most critical business operations for three days. They stranded 70,000 of their passengers across airports across the UK. And the CEO of IHG, the holding group for British Airways, said it was not a systems failure, it was not a technology failure, it was a power failure. In my mind, it's none of

those; it's a business failure of the top magnitude. So, I think these are just examples in my mind where we continue to make the mistakes of viewing these sorts of problems as IT problems as opposed to really grasping and holding them and saying this is a top-level business problem.

John: Yeah it's how you think about the business value. I think in the Equifax example, the CEO said that it was a systems security person that made the error. It was a power outage for the airline. Again, I think if you look at the earlier airline story, you think, if you start looking at all the different disciplines and don't think of IT as this separate thing, which is something we say all the time, IT needs to be part of the business. But we don't really do it well, particularly when a CEO says, "The problem is this person" or "We didn't get electricity." You don't think that way when you...when it's a cargo problem. You look at it as a business problem. And that's part of what we're trying to do—help people understand that high-performing organizations look at the global view of a business, and they don't subset IT excluded from that conversation.

Gene: John, you and I talked earlier about how impactful and meaningful Dr. Goldratt's work has been. Not just *The Goal* but his other books, and more specifically an audio series that he did twenty-plus years after he wrote *The Goal* called *Beyond the Goal*.[11] I probably listened to that audio series probably twenty times during the construction of *The Phoenix Project*, and in my mind, it is the best representation of his lifelong journey as a researcher, as a practitioner. It was probably the best description of all the various tools and techniques he developed throughout his career. And not just in manufacturing but, as we'll discuss later, about how it applies to project management and service delivery, the logical thinking processes, the current reality tree, the future reality tree, all these things that he created. After reading all of his books, without a doubt, it was *Beyond the Goal* that

did the best job in describing what it is, why it's important, how the pieces fit together in the context of his life journey. So, John, tell us about how *Beyond the Goal* affected you and your vision for this audacious project.

John: Yeah, so, like we talked about, I loved reading *The Goal*, I love *The Phoenix Project*, I've read a lot, I think almost everything Goldratt has written. But I think the thing about *Beyond the Goal* that hit me was you could see the path of his learning. You could start off with his story of *The Goal* and that narrative, and by the time he gets to the twenty-plus years, you're hearing this man talk about the delta of what he'd learned. He even expresses questions from the audience—"How many people read *The Goal*?"—and the point where he says, "How many people have read *The Goal*?" and then everybody raises their hand. "How many people actually implemented it?" and nobody raises their hand. Then he points to somebody, and he says, "Why?" and he says, "Well, because of my management." Which is a learning, you have these ideas, and so the idea of *Beyond The Phoenix Project* was, this has been a journey for us.

Since we've met, we've had conversations, and I just thought that we have learned so much since *The Phoenix Project*, even before we started working on *The DevOps Handbook*, the completion of the handbook. Wouldn't it be, in this continuous learning model, to sit down and discuss what we've learned some period after *The Phoenix Project*, very similar to what Eliyahu Goldratt did, and put that together in a body of work to just expand people's knowledge, help companies try to expand on how they become high-performing organizations.

Gene: And I think that was almost two years ago when you approached me with this crazy idea. Since then we've spent hours and hours, scores of hours, working on this, and here we are in the recording studio finally putting this

down. So, it has been so much fun, and I'm looking forward to the many hours ahead sharing what we've learned.

---

† The full quote is: "High-velocity managers are not in place to command, control, berate, intimidate, or evaluate through a contrived set of metrics, but to ensure that their orga-nizations become ever more self-diagnosing and self-improving, skilled at detecting problems, solving them, and multiplying the effect by making the solutions available throughout the organization."[8]

Module 2

Goldratt

John Willis: So, we've talked a lot about *The Goal* and how instrumental it was in the forming of *The Phoenix Project*, and Eliyahu Goldratt, the author. Who was Eliyahu Goldratt? Tell us about his journey.

Gene Kim: From the descriptive perspective, Dr. Eliyahu Goldratt, he was born in 1947. He was a physicist by training. His doctorate was actually in physics, and we're going to talk a lot more about that in terms of how that's a common theme among many of these progenitors of these movements.

It's interesting to find out, how did he end up in manufacturing? It turns out that in the 1980s he was asked by a company to help work around some specific problems around plant force scheduling. And he actually created a software product called Optimized Production Technology, OPT. Some of you might even remember the name of this company. It's sort of a contemporary of companies like Manugistics and so forth. It turns out that this problem obviously captivated him. In fact, he became fixated on why certain manufacturing plants so significantly outperform others. In fact,

there's even some literature that says he was fixated on the problem of why manufacturing plant capacity was so wasted. Why couldn't we elevate productivity to levels that he knew was possible?

That's when he threw himself into a project called *The Goal*. It took thirteen months to write. It was finally published in 1984. Clearly, from listening to *Beyond the Goal*, it becomes clear that this became the journey of his life. This area of study first started out in plant manufacturing. He then created a whole set of techniques and patterns in terms of how to do plant force scheduling and with it how to apply this to project management to service delivery and so forth.

John: Gene, I remember, as we've discussed, you're the one who introduced me to *The Goal*. I immediately fell in love with this man and the way he thought. I started reading all his books, and I remember there's a time I called you on the phone "Gene, can you make an introduction to him?" and you're like, "Sorry, John, he's already passed away." But then you did tell me that you actually had a phone call with him.

Gene: Yeah. In fact, that was in 2004, when my co-author, Keven Behr, and I, we sent an email to him and asked him some questions about his benchmarking journey. It was amazing that he replied. We scheduled a phone call, and it was a ninety-minute phone call I'll never forget. In many ways it was like being Alex in *The Goal*. He had a very specific way of talking and a specific way of asking questions that was unmistakable. It was something that I think about a lot and certainly had a very formative contribution to where we went from there.

John: That's so amazing.

Gene: By the way, I actually saw him again many times at the TOCICO conference. This is the Theory of Constraints International Certification Organization. He was actually running an annual conference where all the consultants and all the practitioners would gather, and you would see these amazing case studies. The man was a very powerful personality.

John: One thing Goldratt is commonly known for is the idea of identifying bottlenecks. What about Alex, and why are they so important?

Gene: One of my favorite lines from *The Goal*...Oh my gosh, just in the last two hours, how many times have we talked about our favorite lines from *The Goal*? Here's another one of my favorites. This is a phrase that Goldratt asserts in any flow of work, there's a direction of flow. But there's one and only one bottleneck. And any improvement not made at the bottleneck is an illusion.[1] He gives us very elegant proof for this. He asserts if you make an improvement after the bottleneck, you'll always remain starved for work, because always waiting for work from the bottleneck. And if you improve something before the bottleneck, the work just piles up even more quickly at the bottleneck.[2] So, I think this is just another one of those amazing logical proofs that he gives, easy for him as a thought experiment.

John: Having said all that, once you know the constraint, what'd it go and say about having to actually manage it?

Gene: It's funny—apparently not enough. It turns out that in the first edition of *The Goal*, as many people have observed, Goldratt...he did something that other people couldn't replicate. They actually rewrote portions of *The Goa*l in a second edition to introduce what he called the Five Focusing Steps. I think the more modern treatment of that is...we would call this the Constraints Management Portion of Theory of Restraints.

What you see in the second edition of *The Goal* and later is the five specific steps that Alex Rogo goes through to magically find the constraints and figure out what to do once you know about them. The first is identify the constraint. The second is then decide how to exploit the system's constraint. The third is then subordinating everything else to the above decision. I think this is so interesting, because this is where Goldratt starts introducing the notion that a minute lost on the constraint is a minute lost to the entire system. When the heat treat ovens go down in *The Goal*, it's not $12 an hour that you're paying the worker, it's actually $20,000 an hour because it's basically like every worker is now idle, showing just how important managing that constraint is. [Step] four is then elevate the system of constraints, and five is repeat as needed.

When I took the course at Washington State University, there was an entire half semester dedicated to applying these steps to different domains, such as plant manufacturing, project management, service delivery.

In the technology value stream, it always amazes me how the pattern of which the constraints are broken are pretty similar. The first place where the constraint usually resides, especially for traditional technology organizations, is environment creation. I think this is something that resonates with anyone who's been in IT operations, especially ones with shared service spanning hundreds of business units or development groups. We never have an environment when we need one. Because they take forty-six weeks to create, every time we want to get a test environment or production environment to stage environment, we have to wait weeks and weeks and weeks. The countermeasure there is, obviously, how do we ideally create environments on demand without having to open up forty-six different tickets, without having to wait weeks. That's the first constraint.

John: I think this falls really well into a lot of the things that we describe when we talk about DevOps writing. DevOps, we talk about automation, and we talk about trying to...the influences of Lean and waste. To your point then, looking at it from the bottleneck analysis, how do we create these environments? Whether it's cloud or it's physical servers, and being able to understand the before and after crosstalk buildup.

Gene: Treating infrastructure as code. It's clearly one of the ways that we've diminished this constraint. After that, the constraint typically becomes code deployment. We can't do multiple deployments per day if a code deployment takes six weeks. In fact, one of the most formative moments for me was in 2008. It was a $2 billion-a-year organization in the ad tech business. Their deployments were six weeks because every deployment required 1,300 steps. Clearly, the countermeasure there is applied—mainly the same technology we used in the first environment creation, diminishing to here. We want to automate as much as possible, and where we can't automate, we want to reduce the number of handoffs so that we can get as close to single piece flow as possible.

John: Here again, what we're looking at in first case is waste. In deployments where we are...multiple teams waiting to deploy something because one group owns this. We talk about continuous delivery as a model that tries to help us enable that. Again, I think that the brilliance, if you will, of this bottleneck analysis is looking at it as a global view, as opposed to just, "Let's do this, and we can fix it."

Gene: In fact, in *The Phoenix Project*, one of the key scenes is a hundred people waiting for their deployment and then it blowing up. That's clearly an example of deployment going terribly wrong.

What I find so amazing is that as an organization goes from code deployment lead times that are measured in months—maybe even quarters—down to minutes, the constraint moves in some pretty predictable ways. In fact, let's call them five phases. The first place where the constraint almost invariably resides, especially for traditional IT organizations that have shared operations, a shared service, is environment creation. We can never get enough of them, and whenever we really need one, we still have to wait forty weeks. That's because the whole process of creating an environment, which is defined in *The DevOps Handbook*, it's everything except for the application—that's database, storage, OS, networking, firewall rules—requires work from perhaps fifty different teams, so that means fifty different queues.

So, the countermeasure to diminish that constraint is that we have to automate that as much as possible. Use techniques like infrastructure as code so that we can ideally on-demand create an environment entirely self service.

John: Yeah, I think we talked about the timing and conversions right about this time 2008, '09, and '10. We saw this surge of this ability to write the kind of code that builds your server, your patchy server, your Java stack, and the whole infrastructure as code—things like Chef and Puppet and today Ansible, and then the idea that the ideal environments we got cloud, we have virtualization environments, and so all this stuff, the automation part just fell really well into place.

Gene: And then the outcomes are amazing. It's that now increasingly, especially with things like cloud, we all have this expectation that we can run a command line script click somewhere and have an environment within seconds. That's something that is still not possible in many complex larger

organizations. So, after we create environments available on demand, the constraint then typically moves to code deployment.

One of the most formative experiences for me was in 2008. It was for a multibillion-dollar ad tech business, and their deployments would take six weeks. It involved something like 1,400 different steps. It would tie up 300 people every time they did a deployment. It was tedious, error prone, and the production outcomes where usually somewhat catastrophic. And so, we can't get to multiple deployments a day if every deployment looks like that. So the countermeasure is, how do we automate as many steps of that deployment as possible so we can ideally go from [1,400] steps to ideally one step? And if we can't do one step well, at least how do we reduce that to the minimum number possible so that we can get as close to single piece flow as possible?

John: And when we talk about single piece flow, and if we think about *The Phoenix Project* and the constraints and the novel version of that solution, then we start working on *The DevOps Handbook*. And one of the primary themes that you get out of the handbook is this notion of continuous delivery.

Gene: Right, in fact that really represents the single piece flow—almost the theoretical ideal where every line of code results in a production deployment.

John: You know I love the Mary Poppendieck quote, "How long does it take you to get one line of code through the system?"

Gene: Right, and by the way, just to maybe further punctuate that, in *The Phoenix Project* the key scene is the horrendous deployment of Project Phoenix where hundreds of people were bracing for the deployment and

then spent two weeks recovering from that catastrophic deployment. So, ideally, we are deploying all the time in single piece flow safely, securely, and reliably.

John: And you know one of my—we talked about Scott Prugh at CSG. One of my favorite scenes in one of his presentations is he shows the before and after of a deploy. In the before everyone's in the NOC [network operations center] room, and they're all frantic. After they go through this transformation of how they get to continuous delivery in their model, they're all just basically watching their cell phones, and its really a non-event for a major deploy.

Gene: Absolutely. So that definitely represents the diminishing of deployment as a constraint. After we do that, then the constraint typically moves to testing. And so, we can't do multiple deployments per day if every time we want to do a deployment we have to wait six weeks for all the testing activities to finish, and often that could be weeks of manual regression testing. It could be waiting weeks to get access to an integration test environment so that we can test our components alongside of all the other components that we are dependent upon. The countermeasure there is often massively automating the test process. Reducing reliance from integration testing to further upstream so that shifting left...so it's more relying on unit testing as opposed to integration testing. In fact, we also probably have to massively paralyze our testing so that our test rate can keep up with the ideal deployment rate.

John: So, when we think about the bottlenecks right in these first three ideas, with the environments, deployments, and tasks, one of the things that I've learned [is] this idea of how do you diminish the bottlenecks. I think

you talk a lot about trunk deploys and how do you get there, and I think you're saying you need all three of these things.

Gene: Oh, for sure. In fact, the only way to make trunk-based developments safe is to have this effective automated test suite that we can rely upon so that we can get to our theoretical ideal where every time a developer checks in code into trunk it can be safely deployed into production knowing that it's going to operate as designed.

So, after that the typical constraint then becomes architecture. I really love this one because the symptom is that every time that we want to make a small little change, we have to get permission from fifty other different people, whether its product steering committees, architecture review boards, chained advisory boards. And the reason we need those approvals is that everyone is petrified, afraid that whenever I make a small change it could cause something truly calamitous to happen. So, what that's a symptom of is a tightly coupled architecture where small changes can result in global catastrophes. So, the countermeasure is that we need an architecture that is more loosely coupled, that allows small teams to be able to independently develop, test, and deploy where small changes cause small failures not large global failures.

John: It's interesting because if we look at, again, convergence, we could go back to what people would call the service-oriented architecture, domain-driven development. Today the large discussion of microservices and then even cloud native, which is this decoupling which just fits really well into this discussion of a potential bottleneck of architecture.

Gene: Absolutely.

John: Another interesting area that came to me is the...I have to go all the way back to the early days of infrastructures code and another—we've mentioned Andrew Shafer, a friend of IT Revolution, he had this great thing where he called it the meat cloud. And what he was describing in the meat cloud was all the toil of building environments and the countermeasures were things life Chef and Puppet...and Andrew was one of the founders of Puppet. And along that time, there was a lot of discussion about ratios. You always like to mention an article I wrote called "The Math to Meat Ratio,"[3] where we used to think about how many sysadmins to how many servers did you have, and in the old pre-DevOps days we weren't thinking this way of value stream. At least in IT we would have really bad numbers, like one sysadmin to twenty servers. Then all of a sudden, infrastructure as code and automation came in, and you started seeing these numbers of one sysadmin to 100 or 300 or, you know...the numbers you can't even...Even four years ago, you'd hear Facebook has 1 to 10,000 or 1 to 15,000.

I love the Instagram story. There's a story of Instagram, before they got acquired by Facebook, about how they started out with this Instagram idea. They had a couple of developers, and they went ahead and allocated somebody as their sysadmin. Then they grew to some phenomenal number of users, and they're "You know, I think we're gonna need another sysadmin." I mean, I don't know exact numbers, but we're getting close to 100 million, 200 million users. Then they show this arc where they grow the development team like seven, eight, nine, and they're still at two—

Gene: Two system administrators.

John: Right. Exactly. It tells this story that if you took this approach, the environments, deployment, and testing, and put that in a global view...He didn't talk about theory constraints—this was the founder—but you got to see this beautiful arc of how if you thought this way, his chart of growth of

customers and servers and infrastructure, it wasn't until, I think, they got to just south of a billion before they figured, "You know what, we should just add a third system administrator."

Gene: And so, Facebook bought Instagram for a billion dollars, and at that point I think they had ten engineers, so Dev plus Ops. I think an even, a more startling example of that—just fast forward a couple years—is Pokémon Go. The fastest property to a billion users...that was twenty-five engineers. So, I think that paints what's possible these days. We should be able to take twenty-five engineers, put them into any modern business context, and achieve Pokémon Go-like things. And we can do that if we have the right architecture.

John: Yeah, and again, I think the key point is...I mean, we love automation. I love automation. My career has been Chef and Docker and all these things. But the real magic happens not just because of the automation, it's because of the analysis of the value stream from waste and, more importantly, the ability to look at bottlenecks from a global optimization.

Gene: Which I think makes it so interesting where the bottleneck ends up. And you hear this from—I first heard this from Adrian Cockcroft at Netflix, and Roy Rappaport at Netflix too. He [Adrian] said, "Then the constraint ends up being the product owners and how many ideas we can come up with." In other words, how many good ideas are actually worth testing with real live customers? He said the bottleneck shouldn't be Dev or QA or Ops or security. That's [not] where we want the bottleneck. And I think that really goes very well with the thesis that Eric Ries and Steve Blank said: the bottleneck should be the creation of good ideas.

John: Right, and be able to—I mean, the original metaphorical picture of DevOps was what we used to call "the aha to the ka-ching." Right? The

moment that somebody has an idea and puts it on a sticky on a whiteboard to the moment that some customer is enjoying or paying for that idea. How fast can you get that pipeline or value stream delivered?

Gene: Ronny Kohavi—we wrote in *The DevOps Handbook*—he ran the experimentation platform at Amazon for many years and is now at Microsoft. He put out this amazing study that said one out of three ideas actually create value. One out of three is neutral, and one out of three detract value.[4] The question becomes, which ones are the ones that create value? You can only find out by testing with real-life customers.

John: So, what we're hinting at is that a physicist and how they apply scientific thinking might actually be a key idea here.

Gene: I couldn't agree more.

John: So, in *The Goal*, we had the automated robots and the enclaves. In *The Phoenix Project*, we had basically a sysadmin and a Java stack. So, this Brent character. He's a great example of this discussion.

Gene: In fact, I think one of the unexpected delights that came out of *The Phoenix Project* is just how deeply the character Brent resonated with the DevOps community, because I think we've either all known one, or more likely, we've all been one. The characteristic of Brent is that no work can get done without him and no outage can be solved without him as well. In fact, there is also this secret sort of feeling that Brent is the cause of the outages too. So, he's this single person that we're all reliant upon for the most strategic objectives, for the most tactical firefighting. It really is, in many ways, the constraint, just like the heat treat oven operator. If Brent is sick, it's not like one person is sick, it's like the entire technology staff is immobilized.

John: It's funny, the Brent character. I remember we had, very early in, this kanban for DevOps.

Gene: Yeah right. That's 2011 with Dominica DeGrandis.

John: Right. And I remember one of the students in the class just started going, "Well, there's the Herbie." And I am "What in the heck is she talking about?" And I had already read *The Goal*, but I didn't tie together the little quick reference of this character.

In *The Goal*, one of my favorite scenes is when Alex is frustrated about what is going on with the plant and all. He also has this, "Oh, I have to do this scout hike with these kids." And what happens is, the kids over the hike by the end of the day are so spread out. He sits there on that evening, and he starts thinking about what's going on, and there's this young boy called Herbie. He's got the biggest backpack; he's the slowest of the gang. He [Alex] sits down and starts playing this little game where he starts figuring out how he could move Herbie around to try to create a tighter variation of the tribe spread. And what that becomes is this really meaningful and important concept that comes out in Theory of Constraints, is something called drum-buffer-rope. I would love for you to explain the drum-buffer-rope concept.

Gene: Yeah, it's another very concrete technique that Goldratt developed essentially to schedule work around the constraints. So, the drum-buffer-rope, let's break that down. The drum is really the notion that the tempo of work, especially the job release function in a manufacturing plant, is really dictated by the completion of work at the constraint. In other words, we only release more work onto the plant floor when the constraint completes the job.

Let's go with the rope first. The rope suggests that its really a pull system, so that any time the constraint finishes the job, there's a virtual rope that

pulls the next job onto the plant floor. As opposed to what typically might happen is you'll release more jobs and they just all pile up at the constraint.

The buffer is really interesting. I had no appreciation of this until we had done a bunch of exercises in plant floor scheduling. So, the buffer serves a very specific role; it's really to protect the constraint. So, the notion of a minute lost of constraint is really a minute lost of the entire system. So, what happens if the work center before the constraint goes down and now we have no job for the constraint to work on? We've essentially starved the constraint, and so the countermeasure there is to slightly overproduce so that we can have a buffer of work before the constraint. So that if certain work centers go down, there is something for the constraint to work on. So that's drum-buffer-rope.

John: And going back to Goldratt using the novel format of taking something very simple, like the Boy Scout in this young man Herbie, and the picture that you see now, if you Wikipedia drum-buffer-rope, is these Boy Scouts and this Herbie somewhere, like a third of the way in, with a rope and somebody with a drumbeat. And it shows how he, the next day, it all worked.

Gene: And to dive into there a little deeper, so what Alex did on that Boy Scout trip is basically lighten the load of Herbie, distribute it across the entire Boy Scout troop. And his realization was that we are only as fast as Herbie. So, whatever we can do to speed up Herbie is what we must do.

John: And then, just to kinda bring it all back together, what we did, or what you did, in *The Phoenix Project* with Brent, was the drive of how the Phoenix Project ultimately started to get successful.

Gene: Yeah. In fact, it's so...sort of rethink through what that team did, once they identified Brent as a constraint, is that they realized that Brent was getting way too many phone calls and essentially being bullied into doing work that he shouldn't do. Because we know that the Phoenix Project was actually the most important thing. And no matter how loud someone yelled, no matter who was calling, they [Brent] should ignore them. They should route that to, in this case, it was Bill and his sidekicks, Wes and Patty. Then, the notion of taking Brent entirely out of outage resolution. He [Bill] just said, "Hey, look. Every time we allow Brent to fix an outage, Brent gets a little smarter and the entire system gets a little dumber." The notion that these are the behaviors that increase our dependence on Brent.

Sounds like there's one interpretation where you can say this is actually very demeaning to Brent. You know, we're taking away Brent's special place in the organization. But I think in reality, when we go through this type of transformation, Brent actually gets to go on vacation for the first time. Brent is actually based on a real person. And I remember around 2004, the real Brent, he said, "For the first time in my life, I got to go on vacation without a pager." And he cried. It was a very personally meaningful thing. It was an amazing thing to hear.

John: I'm a big fan of this convergence. All the things that are coming together at the same time. I remember early on, Damon [Edwards] and I, together, in one of our podcasts, we went out to Kaching. Which was renamed [inaudible], which by the way, is one of the case studies in Eric Ries's *The Lean Startup*.[5] And we went there. It was so interesting to see their view of flow and total flow, and how they were able to implement. One of the jokes that they would make is that it really didn't matter when people went on vacation. This is 2009, 2008. This idea that even then, we still had this

notion of, well, that person can't leave. They had built an infrastructure where everybody...The value was there really were no Brents.

One of the things I did want to point out is that you had mentioned this, we've heard this, that sometimes people look at Brent as the bad guy, which is so ironic. But as we started understanding how to create value from a value stream, what we we're really saying is Brent became more important because we created him more horizontal in his knowledge in fact, which is a precursor to discussions that happen a lot today, which is this full stack engineer or site reliability engineer. So, in a sense, you were actually predefining the way the world looks now, where people really embrace people looking more like where Brent went.

Gene: I think it really gets to the point that when you are a Brent, like in *The Phoenix Project*, life is not so fun. Right? You can't go on vacation, you're being woken up in the middle of the night all the time, and we are allowing a situation where knowledge is trapped in someone's head as opposed to, in the ideal, it's really in the code, it's integrated into the entire organization, and we're all in rotation so that we never create that single point of failure.

John: Just to add one more story from *The DevOps Handbook*. Ernest Mueller and one of the stories there where they created this team of different people in a DevOps project. They brought in a network person, a sysadmin, a developer, and the goal was for everybody to absorb each other's knowledge. By the end of the project, the network people were developers, and the developers understood networking and database, and again I think that's this story.

Gene: That's the National Instruments story in *The DevOps Handbook*. Another great achievement by our friend Ernest Mueller.

John: What we do is we try to turn Brents into people who create more Brents. Is that correct?

There's some other tools that I think I'll introduce. In fact, in a later work, *It's Not Luck*,[6] he [Goldratt] discusses this current reality tree. Can you explain the current reality tree?

Gene: These are a set of tools that Goldratt introduced in his book *It's Not Luc*k, which he had written right after *The Goa*l. It was interesting, but it got a little bit fuzzy for me. In fact, it wasn't until I had taken that graduate course with Dr. James Holt that it started to fit together for me. Let's talk about current reality tree and its cousin the future reality tree.

There is no doubt that these were very critical for the construction of a book like *The Goal*. The current reality tree is essentially a set of problems which Goldratt calls undesired effects that are specifically linked together in terms of causation. At the very bottom of a current reality tree, you have all these small problems that cause bigger problems to the top of the pyramid, and really the entire existence of the organization is now jeopardized. As Dr. James Holt said, the higher you get up in the tree, the more it sounds like our boss's boss's boss's biggest problems.

The future reality tree is really the inverse of that. In fact, this is very useful to construct because essentially what you do is you take every node in the current reality tree and flip them around so that a statement like, "We have too many outages," becomes, "We have very few outages and our deployments are rapid and reliable." Instead of at the top of the current reality tree you have, "Our organization is losing in the marketplace," we have, "Our organization is winning in the marketplace." In *The Phoenix Project*, it's actually very, very useful to be able to go through the construction, create the future reality tree, because it does make some startling predictions about what the ideal state looks like.

John: It's funny, when I first started understanding this concept of the current reality tree, I went out and tried to find some examples. One of the simplest examples is the car won't start. Why won't the car start? It's because the ignition doesn't work. Why doesn't the ignition work? It's actually the whole electronic system is broke. Ultimately you get to the car is actually in a lake. Finally, why is the car in the lake? It's because they didn't put the emergency brake on. I thought that was...Again, I always like the simple version, but it was an example of a picture of a current reality tree.

Gene: Absolutely. In fact, that goes to the other technique that Goldratt developed, which can be called the categories of legitimate reservation or the logical thinking processes. In the back of *The Goal* of the twentieth anniversary edition there is actually an accounting journal that calls the logical thinking processes, the CLRs, the biggest breakthrough in logic since Aristotle. I think the reason why they would say that is whenever you have an arrow from an undesired effect to another undesired effect in a current reality tree or in a future reality tree, essentially it's a way the CLRs—the categories of legitimate reservation—it really gives you the nine different ways that you can dispute whether that arrow exists or not. Does A really cause B? Does even A or B exist? Existence, causality, alternate causality—a lot of these would show up in a logic class, but this is essentially the level of scrutiny that Goldratt would insist on. And he would take a look at, or scrutinize other people's trees. Again, it was very useful for the construction of the book, but I think the modern academics would say the current reality tree, the future reality tree, is very useful for Goldratt, and it was very useful for writing a book in the style of *The Goal* and giving hints of what the solution looks like. But it is probably not as useful in day-to-day practice.

John: You have some real-life experiences with using this concept of the current reality tree.

Gene: When I made the comment that it's not so useful in daily work, my experience of trying to use them in daily work was not so good. In fact, I remember spending a good chunk of a year building these current reality trees. I think I printed it on eleven-by-fourteen paper, stitched twenty pages together, and then showed it to my boss at Tripwire. He was not impressed. In fact, I think he refused to look at any of the boxes. I think just showing that...to show this to an executive is not actually a practical exercise. In fact, I think this is why academics would say that it has a very certain place. Maybe it is useful in making medicine but maybe not selling the medicine.

I think the big breakthrough that was made in the academic circles was something called the core chronic conflict. I think what this pointed at was that no one seemed to really be able to make good current reality trees that were very useful except Goldratt and a couple of others in his inner circle. One of the reasons is he makes this claim that at the bottom of every good current reality tree is the core conflict, but no one really knew how to make a good core chronic conflict. There is a technique called the evaporating clouds that goes through a very methodical process of how you construct it. It was actually that that really allowed us to create the right current reality tree and that really allowed it to be the basis of *The Phoenix Project*. In other words, in some ways you could say the first 170 pages of *The Phoenix Project* is sort of like a traversal of that current reality tree. I think this would be useful for anyone, any senior leader, especially if they are dealing with spanning different silos or organizational boundaries.

The core chronic conflict for DevOps, I think, will sound very familiar. In order for technology to help our organizations win in the marketplace, we have to respond to urgent business needs. That means we need to be able to ship changes ever more quickly, but we also have to preserve world-class reliability, security, and stability. But that means we can make changes never. Both valid business goals. You have to respond to urgent business needs, but

we also have to preserve security and stability. But they lead to...diametrically those actions make changes more frequently and quickly versus making changes less frequently and more carefully. In some ways that really is the embodiment of Dev. Ship, ship, ship. Whereas Ops was all about preserve stability, which means never ship again.

John: I love the core chronic conflict because if I go back to my first experience with this DevOps thing, which was the original Ghent in 2009, DevOps Days that Patrick Debois started. There you saw this idea of, for the first time, there was a possibility of development and operations actually collaborating. People were having these emergent discussions then. And me spending the prior twenty-five years in this industry of that just toil...the two things just never matched. And then it was basically that same year or just earlier that year...there was at Velocity where we talk about now the famous John Allspaw and Paul Hammond's "10+ Deploys A Day" at Flickr.[7] I remember being in that room and watching that presentation, and I joke that when they were arguing about this that—this is my literary license—that people were throwing up in the back of the room. "You can not do this ten-plus deploys a day to production. That's so horrible."

Gene: It's irresponsible. It's immoral.

John: That's right. And what really he [Allspaw and Hammond] was saying is, get over it. It's about collaboration. But today, we add in these works of Lean and we add in the works of the bottleneck from Goldratt, which we both agree is so important, and we get to now crystallize it. Today when I talk about this, I use the phrase core chronic conflict much more often when I'm trying to describe this, that what really became the heart of Dev and Ops, and as you pointed out, all of the years of our memory muscle being in this business was Dev needed to go faster. The market demanded they go

faster, and the operations were on this other side of "I have to protect the fort." The one other thing that I wanted to mention, because I wanted to get your thoughts about this, is in that same Velocity conference, a non-recorded session was a presentation by Andrew Shafer on basically operations, Agile operations, Agile infrastructure, and he produced that now classic wall.

Gene: The diagram.

John: The diagrams of development throwing something over a brick wall to operations, and he codified this idea of the wall of confusion. So, really, it's great how we can tie this all back into the core chronic conflict was the DevOps question we all got excited about.

Gene: You told me something about the Allspaw-Hammond presentation that I never really made the connection until you pointed it out is that...By the way, John Allspaw was the VP of operations for Flickr and Paul Hammond was his director of development counterpart at Yahoo!. But he said at the Velocity conference, you did have Dev and Ops there, but just never on the stage at the same time. I thought [that] was a very interesting distinction.

John: Yeah. And in fact, it was interesting, as this actually started the discussion about...when I saw that, I said, "We have all these kind of starting, emerging startups, and we're talking about these principles. Boy, I wish we could get the enterprise people at the same conference just to..." Because, one of the things I love is...to me, I was just getting to know John Allspaw at that point. I was actually selling him Chef incidentally at the same time, but he was talking about change management as something that was flashy and new. For fifteen years, I've been working with Telcos [telephone companies] in ITSMF, ITIL, that was just a big already very ingrained principles, but

yeah, the idea of getting all these people talking together in the same conversation at the same conference.

Gene: It was just never seen before. In fact, it was a very iconic slide, DevOps and a big heart around it, that showed up in almost every DevOps presentation for the following five years.

John: You also used the current reality tree in the construction of *The Phoenix Project*.

Gene: Yeah, absolutely. In fact, one of the most amazing opportunities was to be able to work with Dr. James Holt. George Spafford and I, we took a class with him, and I got a lot of coaching from James Holt in terms of creating a proper current reality tree using the core chronic conflict as the basis. I think a couple really big insights came out of it. When you go through and create a properly balanced current reality tree it actually does do a very good job in making sure that you thoroughly explore and mention all the right problems. It actually revealed some insights about where problems should be that we could then confirm in the field, and that was super useful.

But then to turn that into a future reality tree, it really does show that if you've gone through all this work to describe all the problems, it forces you to be able to describe all the benefits that DevOps should be able to create. That was also a very useful thing that we could then weave into the story to show how DevOps really is great for Ops and Dev, and the organizations that we serve.

John: In *Beyond the Goal*, there's a really good story about Black & Decker. I love the way you explain it.

Gene: I also loved that segment in *Beyond the Goal*. I think it's being increasingly written about in the literature, but it's this interesting confession that Goldratt makes that said one of the big mysteries in his time was why Black & Decker—which was one of the early and most successful adopters of MRP, manufacturing resource planning software—why were they able to get all these breakthroughs in lead times and project due date performance that very few other people were able to replicate.

To boil it down, essentially what Goldratt says his discovery was is that calculating the bill of materials and the routings was a very computationally intensive task. In fact, it was all done manually by twenty people to be able to match customer orders to do the actual bill of materials generation, order of parts from suppliers, and it was so intensive that they would only do it once a month. It turns out that Black & Decker was actually one of the few manufacturing plants that once they automated that process, they didn't do it once a month, they did it everyday. So, they were able to actually recalculate their entire shipping schedule in ways that they could win in the marketplace that no one else could.

John: I think from what I remember of that story, the first thing I thought about was this idea of cargo culting of companies because he points out why nobody else or other people couldn't succeed. If we look at the history of DevOps going back seven or eight years from the name, if you will, the coined name, I've watched people look at Etsy, and they say, "Oh, if we just take their automation and we do the rituals that they do, we'll be fine." I think the thing that we've covered very well over the last couple of sections is it's your unique value stream that you need to understand and then it's the tools that you apply into that value stream.

What Goldratt was saying, and I think our experience in working with all these companies is, our messaging is don't just look at an organization like

CSG with Scott Prugh and say, "I'm going to do what they did," or look at Etsy or look at...You have to actually go back and do things like the Theory of Constraints, current reality tree, Lean waste things, and get all that on top of your value stream, and then you'll get to these type of outcomes.

Gene: Right and I think Mike Rother, who we'll talk about later, in his book *Toyota Kata*, I think that was what he was very riveted on is how do we get away from just copying the motions and the actions, and, as he says, what color were the kanban cards or what color were the Andon cords? What are the beliefs and the behaviors behind those mechanisms? Goldratt, in terms of the Black & Decker story, he actually came up with some very provocative questions about constraints and policies, and how they pertain to constraints. I'll just read these four questions. He said question number one is, what is the real power of the computer system technology? In his case, it was MRP software. Two is, what limitation does this technology diminish? Three is, what rules help accommodate the limitation? Four is, what are other rules that should be used now?

These four questions are meant to illuminate this problem that the technology—the MRP of technology—the limitation was actually the calculation, the sheer amount of effort required to actually create the plant schedule for the month. Once you diminish that limitation, the new rules should be run once a week or maybe even once a day. This was Goldratt really saying that whenever you diminish constraint, we actually have to zoom way back and say, "What are the rules, policies, rituals we go through that we no longer have to do?" To really get the benefit of the new technology, we have to rewrite the rules, and there are so many examples of rules that we may have outlived in the DevOps world.

John: And again, I keep remembering why I love *Beyond the Goal* so much. This whole section of these rules is...and I think as we have this discussion,

we're trying to take these principles from manufacturing and move them and map them into software service delivery. Some people get pushback and say, "It's two different things." If you look at his second rule of what limitation does the technology diminish in our world? It's not just technology. It is how do we practice? What are the principles and practices they apply?

I think the other thing that I've always been fascinated by on this checklist, if you will, as a thinking process, is that there's always the workarounds. So, you have to be really careful when you look at what limitation does this technology diminish? What are the rules that help accommodate this limitation? He makes a good example of what you're going to have is lots of people who have bypassed. They went ahead and said, "We can't do this. I'm going to add something else." You have to make sure that you include in all those kind of bypass.

Gene: In fact, think about how many people whose job it is to enforce these rules. I'm thinking about when production deployments were so risky; that's why we deploy once a year. Now we have entire change management bureaucracies that really make sure that only the most important changes make it through. Anyone trying to do daily deployments, they're going to run smack into that. I think that's a great example.

Another one that I just love is even the way we budget for hardware procurement. If we want to get the best prices for hardware, we order once a year so we can get the best possible pricing from our suppliers. A lot of infrastructure people, this is the way they were trained, and so in a world where we need to flex capacity up and down or we need to maximize developer productivity, this is the antithesis of how we can actually help developers.

John: This brings us to something that Goldratt has inspired in me. It starts with *Beyond the Goal*, his references to him being a physicist, and [Edwards] Deming being a physicist, but more importantly, his [Goldratt's] paper, which is called "Standing on the Shoulders of Giants,"[8] where he looks back and he actually mentions Deming there, but he mentions Taiichi Ohno from Toyota, and he really gives tribute to how did he get to all these ideas. Goldratt in his purest sense was able to say, "This is not just me," and that's the point of "Shoulders of Giants," but you have an interesting story about "Shoulders of Giants."

Gene: I love that paper where he really does tribute everything that came before him, but I also have this other memory that I think about quite a bit. It was 2007, and I'm at the Theory of Constraints conference. Goldratt's up on stage, and the nature was very much about how TOC has the answers and Lean did not. He went off to give some examples, and one of them was only TOC really understands the power of constraints and bottlenecks. The person sitting next to me, he was director of quality from a large hard disk manufacturer, and he bristled. I could see him just jolt upright, and actually I think he was red in the face. I was talking to him afterwards, and he said, "I'm TOC certified. I'm also a Lean practitioner. We understand constraints and Lean. We just call it the pacemaker. We understand constraints."

I talked to him for the rest of the conference, and it was just interesting to me that there was something about the TOC community that was more inwardly focused. Often, it was us against them, and you can actually see that impact in the TOC community. I think that's something that's very different than we see in the DevOps community.

John: Yeah. In fact, when I was becoming a student of all this work...I didn't study this in engineering school. I started studying after I met you.

Gene: At the young age of fifty-eight.

John: At only fifty-eight. A little younger than that, but maybe fifty, but what was interesting is, to me, it was so clear. I think that DevOps was, we were, in a sense—I'll say for myself—naïve, because we thought we were creating something new. I don't think in [the] early days people realized [that] what we were really talking about came from all these "Shoulder[s] of Giants." I remember having that conversation with you where you told me that there was this tension between Theory of Constraints and Lean people. How could that be? I think that embodies the beauty of the DevOps.

Now, I got into DevOps at an older age, and most of the people I was working with were very young, but people were not looking for silos, or we were...In fact, one of the things I think was interesting is people have always asked for a rote definition of DevOps. One of the things that Patrick Debois, who we consider sort of the godfather, I think he crafted very well, somewhat purposely but somewhat accidentally, is never to put boundaries on it. I think the reason we still don't really have a standard, one-sentence definition was it enabled this movement to embrace...If we had a strict definition, somebody would say, "You cannot include Lean. You could not include this," and you could have those divisions, but that's the thing I think I love most about DevOps.

Gene: I'll make this one observation just to show you how much I agree with that. There [is] no community I've seen that has been so voracious and eager to borrow ideas from different domains and incorporate them into how we do work. That's why I'm just so optimistic about DevOps and the DevOps community.

John: Then just this work that we're producing...you'll get to see we're going to explore a lot of different areas that will naturally fit into these different

principles of learning, learning organizations, and all that throughout the rest of the work of this project.

Module 3

Deming

Gene Kim: John, I remember the first time I saw you give an entire presentation about Deming. It was Puppet Conf. 2013, and I know for a fact this was a presentation that you spent a ton of time preparing for and researching. Let's start with the obvious question. Why do you believe Dr. W. Edwards Deming made DevOps possible?

John Willis: So I remember that presentation, and really, I was inspired mostly by a good friend of ours, Ben Rockwood over at Chef.

Gene: Then he became VP of operations at Joint.

John: Yes. That's correct. He'd done a presentation years back, "The Transformation of DevOps." But I think it was 2011, we had an open spaces at a DevOps Days, and it was called the Theory of Constraints where we were just gonna all talk about Theory of Constraints. I had been reading a lot of the material, *The Goal* and *Beyond the Goal*, and I came in guns a-blazin' of we're gonna have some fun here. About a third of the way through the

session, Ben gives me the time out, and he says, "You know, by the way, John, all this goes back to Deming." The floor dropped on me. I was like, *How could [I have] all this knowledge, and I'm missing a big piece?*

I went back and started reading Deming and trying to understand him. As I was going through that, one of the things that I think stuck out with me was *Beyond the Goal*. We talked about *Beyond the Goal* and how we listened to it multiple times. I think I had listened to it another time after that, and that poignant statement where Goldratt says, "Oh, by the way, I started out as a physicist, and Deming started out as a physicist." It really started me thinking about how there was something different in what these two—first physicists, then management consultants—and why they were saying things. That was my journey.

Gene: I know that Dr. Deming is a very important figure to you, John. In fact, I think you've had as your Twitter avatar a picture of Deming since 2014. Tell us who Dr. Deming was.

John: Yeah as part of that research I actually fell in love with his body of work. To summarize Deming in one answer is very difficult. I think if I had to say, I would say he was one of the most important figures in the twentieth century when it comes to the quality movement. His impact on things like Lean and Six Sigma, just quality in general—you can see it all over the place. And like I've said, DevOps is his influence on just what we modernly look at today as DevOps and the practice principles.

The key thing was that, like I said earlier, him being a physicist. That's the thing that really kept driving me. I started having this theory that the fact [that] he was physicist, Goldratt said this, made this point...And even Goldratt goes on to be...how physicists think differently. I was thinking, when we first saw DevOps, it was...there was something different than everything we knew or experienced. I think there was some nugget there.

Gene: This is great. So, Dr. Mark Burgess, of course I read his seminal paper about CFEngine[1] in 2004. I didn't realize that he got his PhD as a physicist.

John: This led me to realize that Deming was basically studying in the university at a time when science was just changing and transforming from basically Newtonian to quantum. In other words, Newtonian to Einsteinian thinking. At the same time, I had this opportunity that I befriended Mark Burgess, who actually started out as a PhD physicist and then went into computer science. I was able to take my theory of this new science and Newtonian to Einsteinian as a model for why it's influencing the way we think today. He was a great sounding board, because I could ask him these questions and he was able to help me, [to] drive me in the right direction.

Gene: That's amazing. In fact, I didn't know that about Mark Burgess. Most of us know him as the inventor of CFEngine and the famous LISA paper.[2] John, I have to admit, when I first heard you talk about quantum physics as it related to Deming, you lost me. Why is it so important to talk about this transition from Newtonian to Einsteinian? Tell me more about that.

John: Yeah. I think that the trick really is you have to go all the way back to actually Darwin. With his *Origin of Species*, actually, in 1859. I can't understate the importance of what he did to change the way people think and the impact on sciences. You think about, at that time, the world was very deterministic. We felt that things were predetermined, and all of a sudden Darwin is introducing this idea that things happen through the Theory of Evolution. He says that traits are passed on by chance. Probability statistics. The things that we felt were facts and we knew...all of a sudden he's creating a world of science where there's uncertainty, and we have to understand and work with that.

Gene: That's so interesting, because when I think about the Newtonian physics, it really is the universe is a machine, it's a clock. It was very deterministic. And it was actually Darwin that put forth a different model. I can totally start to appreciate that there's a sea change happening in physics, at least how we view how the world actually operates, but how does this connect to Deming?

John: I still have to go back a little further. There's a gentleman, Ludwig Boltzmann, who is considered the father of statistical mechanics. He's working on these complex problems with thermodynamics, and he is stuck. He reads some of the work by Darwin, and he's able to translate his work with biology into physics. Basically, so you have this connection now with the science, statistical mechanics. Then, if you keep following this thread, there's a gentleman named Max Planck who figures out a missing piece of Boltzmann's theory. And for those of you, or people who have studied engineering, you probably have heard of the Planck Constant. As I said earlier, I was having this ongoing discussion with Mark Burgess, and I wanted him to help me understand that piece. I love his quote: "What Plank did is he made the unmeasurable, measurable." And that's a theme, again, for uncertainty. We're trying to measure uncertainty.

Just to follow that thread, you get to Einstein, who sees what Planck has done, and it leads Einstein to his first Nobel prize. Now, you truly have made this science arc of going from Newtonian to quantum, and in the early 1920s, anybody who's studying any kind of science is just immersed in this thinking.

Gene: That's incredible. It really is shoulders [of] giants. All of these revolutions that are happening in physics, in the late 1800s, early 1900s, the inspiration came from Darwin.

John: And one of the things, like I said, not me being a...studying physics and those kinds of science, I started learning that there was a way to describe this as determinism versus nondeterminism, like we talked a little about earlier. In layman's terms, we think of determinism as, "I take a rock. I let it go. I know, certainly, it's going to hit the floor." And a nondeterministic view: "Not really sure. I have to take some probabilities and measurement." As I use the term nondeterministic throughout this, that's really what I'm talking about.

Gene: To make that concrete, we go from certainty and deterministic systems to uncertainty, where quantum effects start taking place. We have observation, the act of observation, changing potential reality. Subatomic shells, it's not known exactly where the electrons are. This is a strange world that people are now being trained in.

John: As we take this all the way to DevOps, which is what we see today, these are very complex systems. The ability to get us...You asked me, how does this all relate to Deming? I would tell you that Deming's ideas have gotten us to a place where we have no choice but to deal with uncertainty, and there's my theory that he has created a path for us to better absorb uncertainty.

Gene: Okay. For sure, you've convinced me that the 1920s is an incredibly exciting time to be in physics. But at this very same time, the management sciences are being created as well.

John: Yeah. You have this gentleman, Frederick Winslow Taylor, who writes this book called *The Principles of Scientific Management* in 1911,[3] literally describing the science of management. It's incredible about how you work, but to the point we made earlier, that they missed the memo. They're still

driving down this path of very rigid command and control. The structure of what he's trying to do is really driving this efficiency movement.

You know, motion studies, timing, looking at arcs, looking at angles of things. In general, workers are workers, managers are managers. Complete separation and basically just looking at very specific...How deep do you put a shovel in the ground? Although it was a major breakthrough, it was still deterministic, and it really created a thread of determinism in management thinking.

Gene: You know, I may just expand upon that. This is in a time of the Industrial Revolution. You know we are building assembly lines, railroads, telegraphs, and steamships, and so to be able to extract more productivity seems like an amazing thing. So you have a hundred people lined up, and if we can figure out how to displace more dirt by changing the shapes of the shovel or prescribing how deep in the ground we dig, we have those hundreds of people working a day for ten hours a day. That's a material improvement that we are making to the economy.

John: And you know what is interesting now, when you hear people talk about Taylor, they refer to it as something called Taylorism, and in a lot of ways it is actually thought of as a negative. So again, it just creates this point that that model actually worked for efficiencies in the Industrial Revolution, but today is thought about as something that is kind of command and control and certainly from a DevOps perspective is something we want to break out of.

Gene: You had this great observation about anyone who has an iPhone should be thanking Taylor, that he helped make that possible. It's actually enabled prosperity that the world had never seen before.

John: No, absolutely. I think when I hear people say Taylorism, as if it is evil, I mean, quite simply Taylorism is a deterministic way that has created prosperity and wealth, as you describe, and Deming creates a path that is nondeterministic.

Gene: Over the years in our conversations, we keep describing Deming as a person who always seemed to be in the right place at the right time. He's like the Forrest Gump of both physics and management sciences. I mean, he is everywhere. So, let's go back to Deming's life. Right now, where is he? What is he doing?

John: I love the Forrest Gump reference. You know, he [Deming] was a smart man, but he did just wind up in really cool places. You know, like we talked about him studying at a time of incredible science transformation. But his first break, if you will, is that he gets an internship at the Weston Electric Hawthorne [Works] factory. And one of the things is that he runs into a mentor there. There's another gentleman doing some incredible stuff, called Walter Shewhart. And Walter Shewhart is an engineer who's working on quality issues with telecommunications, the whole from phones to switches to lines. It turns out that he's a scientist as well, and he's trying to apply some of these kinds of models of probability and statistics and, in fact, Walter Shewhart, one of the first things he's known for is creating this thing called statistical process control.

And so, Deming is there at this time when he [Shewhart] is inventing this paper about something now that's used in industrial control systems all around the world, and Deming happens to be there. Yeah, that's his first great opportunity.

Gene: So this factory. In hindsight, it makes total sense why they're working on these problems, but this is a place that did make just telephone handsets.

It's about the telephony infrastructure, like switches and cables. This is the place where they would install boxes in the ground, and if something went wrong, they'd have to figure out what went wrong and where. I mean across thousands of lines of cable, so the outage cost of getting things wrong was indescribably huge.

John: And you know, again, I think that another thing, if you look at Deming's career, I think a lot of his discussion was around statistics and variation. We'll cover some of that later, but you can't emphasize how much the influence of what Shewhart was trying to tell people about using statistics and variation as a better way to understand large problems that you could just, again, deterministically say, "Okay, this is my number of defects." You had to come up with a smarter way to understand defects.

Gene: In fact, this whole Hawthorne factory is a pretty interesting place in its own right. There's something that's called the Hawthorne effect that was actually named after the plant, John?

John: Yeah, it is. You're studying, you're researching, you find this other article that has really nothing to do with Deming, but it's interesting that he's at a place where the Hawthorne effect is really the study of people in work and the lighting and all this. And actually, some would say it was, it created the foundation of sociology. So, again, Deming, like—I love your Forrest Gump reference—he just seems, you could picture him almost being photobombed from the father of sociology. So, yes.

Gene: And so, he's working under a scientist named Shewhart, and so Shewhart is famous—well, maybe not famous—for something he created that actually Deming made famous.

John: Yeah, and you know, it's just another thing if you look at what Deming talks about in his influence, a big part of it is statistics. But the second part that he picked up, again, just being there and being able to absorb some of this incredible knowledge being created by Walter Shewhart. Shewhart created something called PDSA, Plan-Do-Study-Act, which at its core is just scientific method or scientific thinking. You plan something, you have a hypothesis, you do it, you study the results, and you take an action, you know? It's the joke in our industry: we're really good at plan-do, plan-do, plan-do.

So, Shewhart basically creates this model. Deming, again, just falls in love with everything about Shewhart. One of the quotes I love [is] that they say that Deming was better at explaining Shewhart than Shewhart. So, at one point, Deming really just changed it to Plan-Do-Check-Act, and now we see Plan-Do-Check-Act in lots of different places, and it all comes accidentally because he takes a summer internship at a telecommunications factory.

Gene: And really, we can trace so much of what we think of to Plan-Do-Check-Act, Plan-Do-Study-Act, whether it's a Lean startup, whether it's a hypothesis-driven development, the whole notion of the scientific method, I mean, is really being...those techniques are being forged at this internship.

John: Absolutely. If we go back to...we talked about the Lean startup, Eric Ries's build-measure-learn, it's another form of basically Plan-Do-Check-Act. And even today, we look at, we really try to push in when we talked about DevOps. Again, always trying to tie why Deming was influential in DevOps. A lot of discussions that you hear today in presentations is being hypothesis-driven and being...making sure that you break things up into little experiments.

Gene: So, Deming had a couple of interesting summer internships. Tell us about his first full-time job.

John: So, he's gets his first job, it happens to be at the Department of Agriculture, and now his influence of his background in sciences, his influence with Shewhart and statistics, he has this opportunity to work on one of the early censuses, and he figures...almost like what Shewhart was doing with telephones "We're doing this wrong. Why can't we add statistical analysis?"

So, he was the first person to actually introduce statistical analysis into the census process, which is actually still used today. So, the thing that I love about the Deming story, beyond just the Deming to DevOps, is that this man is just really an American folk hero that, really, very few people really know his influences.

Gene: You talk about Deming being a hero. I think that's so applicable to his contribution to the US war effort, and that's World War II.

John: Yeah, so if you think about his now passion for statistics that he's learned from Shewhart, now he's gotten under his belt his first real job where he's made significant changes to the way we do [the] census. He starts writing a lot of papers, he gets known in the industry and the actual...During the war, the Secretary of War asked him to come and put together a class to train managers at Stanford University.

So, you have to understand the setting. Which is all of the factory workers and factory managers have gone overseas to fight. So now, all of a sudden, we have this demand to create tanks and jeeps and planes. And we need them to be high quality in a very short period...Does that sound familiar. So, his course is designed to really crash-course people who don't have this experience to put these types of quality initiatives here. So, it's said

that over a two-year period, he trained over two thousand plant managers. And so, who knows what the blast radius of this is, and there's been many stories written about the possibility of the war effort being somewhat contributed to the quality initiative that he created for the equipment, the planes and trains, the planes and tanks. So, in other words, he was a war hero as well.

Gene: And that's just amazing to hear, because without a doubt the US industrialization...that is considered, without a doubt, the most massive industrialization and mobilization the world has ever seen. I mean, that contributed to the US victories on two fronts, and to think that he was essentially building a massive Six Sigma-like program for the entire war effort, like GE did in the 1980s. That's incredible. And so, you would think that all these amazing lessons would then be kept, and you would then find traces of them in the post-World War II economy as well.

John: Not exactly. I mean, you know we talk about change is hard. So all these war heroes come back, of course they get their jobs back, which is right. They just went and defended our...saved the world. Defended our country. They come back, and they're just, they're gonna go back and do exactly what they were doing.

So any remnants of "What was that stuff? Sorry." So no, actually almost everything that he did during this short period was basically lost. There was really no evidence in the US of this initiative, even though it was known to be of significant impact. It's just the nature of...We went right back to a deterministic way of doing things.

Gene: That's incredible and yet totally believable. So, we had this momentary surge in productivity that helped the US fight and win the war. But that

largely disappears as the US goes back to a peacetime economy, which I guess sets us up for what Deming is really most famous for.

John: Yeah. So after the war, part of what the US effort...Actually General MacArthur was tasked to actually send people over to Japan to help build their economy. So, in your Forrest Gump metaphor, he gets sent over there to help them build. And of course, he only knows one way to teach. There was something he said at one point in one of his interviews where he said, "When I got there, I noticed three things." And they always say that Deming loved Japan. He fell in love with the people, the culture, and he said, "I saw these three things. I saw a magnificent workforce, I saw unsurpassed management, and I saw the best statistical abilities." And in his very simplistic way, he said, "All I did was put those three things together to help them invade the US."

And I think he purposefully used the word invade, because again they said he fell in love with the Japanese culture. They had just been decimated, invaded. In fact, there was a whole question about all these scientists during the war effort were doing a lot of heavy statistical analysis, that were now out of work. There was no...In fact, there were treaties that said they couldn't actually work on things. So here they are just sitting there. And this gentleman comes over and starts helping them apply this to now manufacturing. So I think, in his mind, I think the word "invade" meant he was gonna help them beat the US manufacturers, specifically, in the end, some of the large big three auto manufacturers.

So, I think if you look at Deming and his influence, there's another great story where in 1950, it's called the Mount Hawkins Summit, where they said that seventy-five percent of the wealth of Japan is in this summit. And he says to them, "If you follow my ideas, within five years you will be a world economic power." And almost to the day, in five years they do become...It

goes from Japan building shoddy things—equipment, cars—to high-quality, low cost.

And one other thing I think is really interesting is, at one point, he was so recognized by the Japanese economy, the world, again where the US doesn't even know who this man is, really. And he gets this...He's the first non-Japanese person to win what was called the Order of the Sacred Treasure Award. So, in the end, I think you're right. One of the most impactful things he did was going over to Japan, and I've used the phrase, he became the Shakespeare of quality in Japan. But as we'll learn later, really the Shakespeare of quality of really everybody.

Gene: The results of Deming on Japan are pretty evident. It has become one of the world's largest economies. It's known as one of the world's most high-quality manufacturers. And so, I think that certainly makes the case that Deming's ideas had a tremendous impact on Japan.

John, let's switch gears and talk about how Deming has influenced DevOps.

John: You see Deming's influence all over from the beginning of the DevOps movement. You see in DevOps presentations these points that are made, and you can actually tie all these back to a work that he created in 1986 called *Out of the Crisis*.[4] It was his first book. Before then, he had just developed a lot of research papers, but this was his first book. And the significant thing about this book is he defined something...what are now known as his 14 Points. And he created this list of points that are significant.

Gene: John, let's go through each one of Deming's 14 Points and share with us your favorite concrete DevOps connection.

John: Yeah, sure.

Number one, create a constancy of purpose. Deming called this the aim. This was the...Think of a target where you wanted to get to so people could see that as the place that everybody needed to go, this purpose of why we're here. The goal, for example.

Number two, adopt the new philosophy. This is really your lead, not management. Replace the top-down, command-and-control structure. Cooperative leadership models.

Number three, cease dependencies on inspection to achieve quality. We see this all over DevOps. Don't wait until it's done to test it. Build quality in from the start. These are core principles of DevOps, and even our continuous delivery and test-driven development.

Number four, end the practice of awarding business on the basis of a price tag. There's this old saying in the early days of cloud: it's not about bottom-line ROI, it's about top-line ROI. In other words, it's not how much it costs to make it, it's how much money you're gonna make if you make it. We also talk about moving to Agile budgeting. Not creating projects that are budgeted. We get more agile in our delivery. And even our Lean start-up, the MVP model is [a] good example of this.

Number five, improve constantly and forever the system of production and service. This is really our second and third way. For example, our feedback loops and amplifying feedback loops using PDCA. Later we'll talk about the Andon cord. The ability...We see this stop the line, break the build, and Devs wearing pagers. The emphasis of delivering a service with test-driven development, behavior-driven development.

Number six, institute training on the job. Basically, what we want to do is create a system for thinking. We talked about global optimization earlier in the Goldratt section. And even if we look at something like Peter Senge's *Fifth Discipline*.[5] This is about learning organizations. These are the things he

was point[ing] out that just completely translate to what we're talking about in DevOps.

Number seven, help people and machines and gadgets do a better job. The thing that I think is interesting about this is Deming saw, over sixty years of his career, this idea that humans and machines had to work together. If you look at a deterministic system, it's a world where a person drives a machine. In complex systems, which really is what our modern IT systems are today, the humans have to be equal parallel actors in the complex system. Deming saw this relationship of, again back to nondeterminism and deterministic, early on. And it really fits in this point.

Eight, drive out fear so that everyone may work efficiently for a company. This is the whole failure. I've said many times, "Failure is the new black in DevOps." And here another...Something that Deming saw very clearly throughout his career. And we see this is in Chaos Monkey, the anti-fragile nature of how we think about things. Blameless postmortems. And even when you think about driving out fear, what we're trying to do is remove cognitive biases. A lot of discussions in DevOps about cognitive biases, blameless postmortems, retrospectives.

Number nine, break down barriers between departments. If we've learned anything since the beginning of DevOps, it's that high-trust cultures are high performing. In fact, the four years of *The State of DevOps Report*, we found that organizational culture is the strongest predictor of IT performance. Cross-functional collaboration. Just shared responsibilities.

Number ten, stop management by slogans. Slogans saying...The slogan slaying, I used to say, the old pre-DevOps, the CEO or the CIO comes in and puts a poster on the wall with a pyramid, and now we're all supposed to figure out what he really means, because slogans are such abstractions. Zero defects, we can never fail. This year we will have no defects.

Number eleven, supervisors must change sheer numbers to quality. So, what Deming is saying is that the supervisors must make workers feel this pride in what they do. Today we talk a lot about people in IT, we have this luxury of being intrinsically motivated. Do we get to work on open source? Do we get to work on projects and DevOps? Do I get to be the CICD [continuous integration/continuous delivery] person? Deming is explaining that supervisors have to create an environment because, when you create that, you get higher levels of quality. It's like intent-driven management, not the man-driven management.

Number twelve, abolishment of the annual or merit rating of management by objectives. Here again, Deming hated MBOs [management by objectives]. You know today we have KPIs [key performance indicators], we have MBOs the cool kids in Silicon Valley now call them OKRs [objective and key results]. And they're very deterministic. Even these new start-ups think that calling them OKRs instead of MBOs, that it's different. But it's okay, it's the beginning of a year, and here's the things that we expect to happen by the end of the year, or the end of the quarter. That's not how complex systems work, and Deming swore on this. So, he didn't like this idea. He believed in allowing emergence to happen.

Number thirteen, institute a vigorous program of education and self-improvement. This is our kaizen with self-improvement or something that was referred to as kata that you get from Japan, which is creating this movement over and over to get perfection. Moving from improvement to an art form.

Number fourteen, the transformation is everybody's job. This goes back to even point one. We all have to have this purpose, this aim, this goal; that's how we transform in a global view. You're not just the security people, you're not just the network people. We have to see this all together as the service that we're delivering for our customer. And again, even though Deming was

creating these 14 Points in his work in 1986, this was the embodiment of sixty years of the way he thought, the way he explained things, and today we look at DevOps and we say...We see a presentation with all of these concepts and we go, "Wow, that's amazing." And that's why when you ask me, "Why is Deming so relevant to DevOps?" I tell you that his body of work over his lifetime proves out these ideas that we are using in DevOps today.

Gene: Wow, John, what can I say? Maybe in the last hour, you have convinced even me that Deming has something to do with DevOps. Now, that's really great. Just maybe to reflect a second, it is just amazing to hear how profoundly Deming understood these things and to what extent that we have incorporated that into everything we see in DevOps, especially when we talk about quote/unquote culture, it incorporates so much of the things that Deming stated so clearly.

John: And I think I could go back to the arc of Deming's Forrest Gumpism. Almost every point there's a human element. And I think that when he went to Japan, it really helped him more understand, you know...The Japanese culture had a very humanistic way of thinking, and with all his...The point was that...DevOps, we always say DevOps is about culture and behavior, or DevOps is a human endeavor that creates performance and quality. And really, that was Deming's lifelong mission to promote this idea of...the human has impact and needs dignity, and for the purpose of getting quality and efficiencies.

Gene: Wow, yeah, it seems like that would be the perfect way to end this module on Deming, but we can't, because there's this amazing other work that Deming did after writing *Out of the Crisis* that we absolutely have to talk about.

John: Yeah, so, in all Deming's work, a lot of academic research and papers, really only two books, and his last book was basically called *New Economics*.[6] It was written in 1993, which happened to be the last year of his life. And in that book, he put together all of his ideas in one, I will call it, simple model or description. He called it the System of Profound Knowledge. And the thing about the System of Profound Knowledge was—and I'll go through the pieces of it—but what he was saying is, it's like you get this aha moment of your life of all the things, and you say okay, all of this looks like complex systems. And so, in the System of Profound Knowledge he was trying to describe a lens, not so dissimilar to what Goldratt was doing with current realities, sort of this lens of how you could look at a complex problem or complex systems.

The System of Profound Knowledge is actually made up of four pieces. One is called the appreciation for system. This is systems thinking. Global optimization, in Goldratt terminology, it's looking at the whole. Lots of work. Again, Deming was a student of science, so all the works that were coming out about systems thinking, he was involved in that.

The second is basically what's called the knowledge of variation. We talked about what he learned from Shewhart, about statistical process control. It was a big part. And one of the things that Deming understood, that most people don't really understand unless they take an operations course, is that...Most people don't understand the real idea of variation. In short, there's what's called common cause and special cause variation, and...they say the deadly sin is not really understanding how to manage one versus the other. If you deal with these anomalies as if they're actually common, you're making mistakes. If you deal with common variation as if they're anomalies, like firing somebody for something the system created. There's a whole deep work there about how to apply variation to that. So, that's a known body of work.

The third piece is what was called the theory of knowledge. So this is the PDCA. This is scientific thinking. This is a scientific method for everything you do. So, here's the thing, those three pieces...I would say, Deming, and Shewhart, and his body of work, but you could say really any great management thinker, could put all that together, create a framework, but you probably wouldn't be able to call it, excuse me the pun, profound.

Gene: *[laughing*]

John: But what, to me, makes this System of Profound Knowledge profound, is the fourth piece to it. And again, he thought of this as a lens. You had to use each almost like a camera lens, where you had to use each of these to understand the full picture. The fourth piece is what's called the knowledge of psychology. The brilliance here is that variations, system thinking, scientific method—all great tools. But what he would say is that if you didn't understand the cognitive nature of how people think and their biases, all that would be just useless. He had these great examples. Actually, people who studied him in hospitals would use this example of a hand-washing campaign: I want everybody in the hospital to wash their hands. But if there was a group of people that believed that that didn't kill germs, and that was their fundamental belief system, then any chart, or variation, or...

Gene: [*laughing*]

John: It wasn't going to help. And so he understood, and he knew...and so if I go back to DevOps, one of the things that Damon Edwards and I had coined is this CAMS—culture automation measurement sharing—and we would always talk about, you have to understand culture. But very rarely did we see any frameworks from anybody who make[s] culture a first-order primitive of their model that includes a lot of other technical things. Deming, again, his whole lifelong body of work put all this together with

what I call that first piece, theory of psychology, the culture aspect of it. And that's why I do think the System of Profound Knowledge is an incredible encapsulation of his life work described in a single way.

Gene: [*laughing*] We were at DevOps Days Detroit together earlier this year, and what occurred to me for the first time was how the knowledge of psychology has been elevated in our awareness, at least for me. And I think the most obvious one is Dr. [Daniel] Kahneman's work around mode one, mode two, thinking fast and slow. And it was that amazing presentation that was given on just how that influences planning and estimation and project management. I guess I had a huge aha moment that said, "Holy cow." That thinking affects so much of what we do—the decisions we make, the shortcuts we make. It just seems again, Deming was way ahead of his time.

John: He had that...in Kahneman's thinking there's the, I think it's called Mary's GPA. And so the story is, Mary was an avid reader from the time she was five. And then you ask the question; she just graduated high school, what do you think her GPA is? And you're, wow, it's 3.7, 3.8, and you've made this incredible leap because they're "Wait, let me tell you about Mary. Mary was an alcoholic, she dropped out of school," right. So, that's what Kahneman was making the point of, is that we tend to let our biases, or our system one or system two, get in the way of what we think. And just to tie that all the way back, Deming put that as an equal lens—system thinking, scientific method, and the study of variation, as an equal pillar or equal lens in this thing he called the System of Profound Knowledge.

Gene: John, you had made this astonishing observation that he actually wrote *New Economics* in 1993 the [year] he died. And you had speculated that his ideas would have gone further had he been around to help promulgate and propagate these ideas. Can you talk a little bit about that?

John: Yeah, I've talked to a lot of people, and I've studied a lot of different areas that have used Deming's ideas, like in healthcare there's been a lot of adoption of his stuff. And more often than not, you'll hear people talk about variations of the PDCA, we can see that's almost everywhere. Statistical process control is ingrained, the ability to understand variation—all those principles...his references of 14 Points, all over.

Very rarely when I talk to somebody and I ask them, even who know Deming, "Have you implemented System [of] Profound Knowledge?" Almost all the time, I get the answer no. And it's almost profound to me why that isn't the case. The only conclusion I have come up with is, people create bodies of work and you either need somebody to be the profit of that or...go back to "Shoulders of Giants," if you don't have somebody take...yourself...if he had another twenty years, maybe, to drive that discussion. I mean he had what, sixty-seven years to drive Shewhart's ideas. He had really no time left on this earth to drive what was his final encapsulation of his brilliant ideas. And at that point, there was really nobody picking up that torch to try to—

Gene: And yet, there's no doubt that Dr. Goldratt knew exactly whose shoulders he was standing upon. So, John, what's the last word on Deming?

John: So, in the beginning of the discussion you asked me who Deming is, and I said it was hard to just answer that in one answer. We walked through his career, his impact, the places that he was at, what he learned, how he was able to build—on each piece of his life he took something with him. He started with science as an educated scientist. He went off and met Shewhart. He then went on to have his first job to take Shewhart's [ideas]… On and on to Japan. And so I think this person was able to just keep learning. And imagine you're, by the way ninety-three. He was ninety-three years old.

Gene: [*laughing*]

John: At ninety-three, he is actively participating in learning. And I think we go back to DevOps. What do we think about DevOps? It's just this constant learning. Why do me and you go to these conferences? Why do we have the DevOps Enterprise Summit? Because we want to continually learn, so I think that's the...I said that I think he's an unknown American hero, but his real story is that he was a voracious, lifelong learner.

Module 4

Lean

John Willis: Out of all the bodies of work, it seems like Lean has been the most influential for understanding DevOps. Tell us why Lean is so important in the DevOps movement.

Gene Kim: Absolutely, and to tell a story about Lean, we first have to talk about Toyota. Dr. Steven Spear is probably one of the most well-known Toyota researchers, and he gave this amazing talk at DevOps Enterprise in 2015. I love the way he said it. He said, "It is impossible to overstate the degree with which Toyota was able to dominate the US industry." He said in 1958, Toyota was only one-eighth as productive as the world standard. By ten years later, it is double the world standard. By 1973, they enter the US market with a low-cost product with unprecedented levels of reliability. By 1985, they're not only the world leader in productivity, but they're upgrading their models twice as frequently as world standard. By 1985, they begin manufacturing in the US for a variety of reasons that we'll talk about later, and by the mid-2000s, they are nearly tied for being the number one world's

largest automaker. It's amazing that Toyota dominates every market they enter. So it's just economy cars in the 1970s, but over the following several decades they become the leader in mid-segment cars and in luxury cars, with the Lexus brand.

John: Yeah, and you know at that point, I think that Toyota was so confident, from what I've understood and researched, that they only introduced the Lexus in the American market at first.

Gene: It's not just that. They entered the hybrid car market with the Prius brand. It's not just about market leadership. Spear points out in his talk that Toyota in 2015 is making nearly $3,000 profit for every vehicle they sell. That's three times higher than the number two, which is Ford, and so all of these profits are able to be plowed back into research and development to make even more competitive products that they can bring into the marketplace. This is the backdrop for what causes this amazing response to Toyota and Japan.

In fact, I think one of the best examples of this is MIT in the late 1980s. Three researchers raised $5 million for an incredibly ambitious five-year project to understand the state of automobile manufacturing and start making projections about how other automakers can regain competitive parity. That's Dr. James Womack, Dr. Daniel Jones, Dr. Daniel Roos. They write this amazing book that's called *The Machine That Changed the World*,[1] released in 1991. I remember reading this book and just being dazzled by...I read this book probably five times after I bought it, and this is just the beginning.

There are scores of researchers who are now poring over Toyota trying to understand, "What are they doing differently, and how can other organizations replicate these amazing outcomes?"

John: The impact that it had on the American culture even made it into Hollywood. It's kind of a joke movie, but the Michael Keaton *Gung Ho* movie, which showed that story where the people in America at that time were thinking about, "We're losing to Japan."

Gene: We're losing to Japan!

John: We are losing a war here, yes.

Gene: So that movie, G*ung Ho,* comes out in 1986, and that's two years after the joint venture between General Motors and Toyota to create the NUMMI [New United Motor Manufacturing Inc.] plant in Fremont, California. This is where the Fremont plant is actually one of the worst-performing plants in North America and within a year becomes the best performing plant, on par with any plant in Japan.

John: One of the interesting things there about the Toyota culture, which we'll cover a little bit later, was that when it was just a General Motors plant, then they actually immersed the same workers in the same plant with this different kind of culture, and all of a sudden it's actually performing with the same people in the same place. That was the most fascinating [thing] to me about the NUMMI story.

Gene: Yeah, there was this amazing *This American Life* podcast about the NUMMI joint venture[2] where they interview workers and managers and union leaders and researchers about that amazing experiment. One of the most moving parts for me was listening to this worker who was there before and after the joint venture. He was tearing up, choking up, as he said his experience at the NUMMI joint venture was the most rewarding and professionally rewarding period of his entire life. It changed people.

I think just to show the economic backdrop, this anxiety that you're pointing at: around 1987, *Time* magazine[3] has on its cover the rise and fall of the great powers showing how, essentially, the 1800s was a century for the UK, that the 1900s the century for America, and that the twenty-first century is going to be dominated by Japan.

The response to this is to mobilize the research community. *The Machine That Changed the World* is just one of many, many books that come out around what Toyota's doing. We have *Learning to See* by John Shook and Mike Rother that comes out in 1999.

John: That book was really instrumental for a lot of us in the DevOps movement, particularly when we were doing consulting and services. That book was a how-to on tool set that was common in Toyota, which was this value stream mapping, where you looked from right to left and tried to work the flow and understand the flow. What was also interesting to me as I talked to people who had been successful, a lot of the DevOps enterprise customers, if I asked them how they got started, almost everyone I talked to will say they started off with value stream mapping. You can't understate the importance of that piece of work for the DevOps community.

Gene: Absolutely, and *Learning to See* was one of the first books that really started codifying these practices that are used in the Toyota production system. That's the same year that Dr. Steven Spear has an article in the *Harvard Business Review* called "Decoding the DNA of the Toyota Production System,"[4] and that is actually based on work that he did in his PhD doctoral thesis at the Harvard Business School for which he actually worked on an assembly line for a Tier 1 supplier for six months.

John: I love that article because one of my favorite quotes from it is that Toyota was a community of scientists continually experimenting, so it just

goes back to...It's not just the impact of what Toyota meant, it's the way they thought where his observation was that they were constantly experimenting, this hypothesis-driven way of thinking about doing work.

Gene: Absolutely. We have *Lean Thinking*[5] coming out in 2003 that's by James Womack and Daniel Jones. *Toyota Way*[6] comes out the same year from Dr. Jeffery Liker. *Toyota Kata*[7] by Mike Rother is written in 2009. That's a book that we know very well in the DevOps community.

John: You recommended to me that book, and the thing I love about that book, again, to understand Toyota is...In the beginning of that book, I think, Mike Rother makes a comment that there's been, I don't know if he says one hundred, but lots of books written about Toyota, but in this book we're going to talk about the hidden side of how Toyota operated. I think that Mike's understanding in getting to the view of basically how there was this culture and understanding which was different than just looking at how they did the tools.

Gene: I know, exactly. And, again, oddly enough, the same year, in 2009, Dr. Steven Spear comes out with his book called *The High-Velocity Edge*.[8] The point here is that the success of Toyota catalyzed an entire community of researchers to mobilize and try to understand what is Toyota doing differently so that other organizations can replicate their amazing outcomes. This is how...and it's all put under the moniker of Lean, so Lean then joins the family of the primary quality movements, so joining the Theory of Constraints that comes from Dr. Eliyahu Goldratt, Dr. Deming and TQM [total quality management], and Six Sigma, which is created in 1986, around the same time. I think that the interesting thing here is that of all these movements, it is really Lean that becomes the most successful. Not just in

just the number of books published, but as you and I have discussed many times, it's probably through Lean that most of us get indoctrinated on our journey to DevOps. John, you and I have often discussed that Lean is without a doubt one of the primary areas that DevOps draws most from.

John: You know, exactly. I couldn't have said it better. Your description of the importance of how we got to Lean and how important it is. As we went through the early history of this DevOps movement, we learned more about value stream, and we learned more about flow, and what we really learned was a lot of our influences either are known or not known, but at some point, it was clear that Lean is the biggest impact and driver. In fact, in *The DevOps Handbook*...a lot of what we talk about in *DevOps Handbook* is the notion of how Lean impacts each one of the case studies. It's just so important. I mean, you can't get out of a conversation today of talking about DevOps without recognizing the influence of Lean.

Gene: John, as a little side note, we spent so much time talking about Deming. You would think that Deming would be at the forefront, at the center, of this economic and research response to Japan.

John: Yeah, no. Actually, there's a really interesting story there. The...in 1980, NBC does this documentary called *If Japan Can...Why Can't We?*[9] It's a response...it's a documentary trying to understand why the American market, particularly the car market, the manufacturing market, is just losing the battle.

And so, a big part of the documentary is about looking at Detroit, what's going on in Detroit, and then in about the last fifteen minutes they have this story about this person, turns out to be Dr. Deming, and how he had worked with a corporation out of New Hampshire to help them improve.

And what's interesting is it's for America, this "Oh, my God," aha moment of the reason, possibly, that the American manufacturers are getting beat by Japan is because an American went over there.

So yeah, actually, at the time Dr. Deming was eighty years old and he was almost retired, living outside of Washington, DC, and was basically unknown in the… I mean, unless you were a deep researcher, you just had no idea, not only who he really was, but [that] he actually created this monster.

Gene: That everyone's mobilizing and reacting to. So, Lean codifies five areas, but I think it's important to note that just because they were documented in the 1980s and the 1990s, some of these practices go all the way back to the beginning of Toyota's company.

John: That's right, Gene. I think to really understand Toyota and their culture, you have to go all the way back to the original founder, Sakichi Toyoda and his routes.

It turns out he is man of little means, his mom is a loom operator, and ultimately...and he's an inventor...his idea is to improve this...the workers...this loom operator.

So, he just continually is inventing. You can imagine at some age he is actually looking at and putting things together, and he's working [with] wood, then he's going to metal, and at some point he actually creates a company.

Gene: The Toyoda Automatic Loom Works company in 1926?

John: That's correct. That's the original Toyota company, which is actually a textile company. And as I said, he keeps inventing, and when you read about him, it turns out you can really see this kind of continuous improvement

idea that he's always...even drawing early, hypothesis-driven way of thinking of just trying something new, seeing if it works, trying something new. And at one point, he's actually probably more famous for this idea of what's called jidoka.

And, basically, that comes from...he had noticed that when the textile needles were running, if a needle broke, it was really a manual observation that somebody had to see. It could have been seconds, maybe even minutes, and what you'd have is a defective textile. And so, he thought about a way he [could] create a weight so that when a needle broke, it would immediately stop. Kind of the original break the build, if you will.

Gene: And, just to make sure I understand. So, the problem that he was trying [to] solve was that the needle would break, and then the yarn would have knots in it, and then you could have yards of fabric that would be defective or maybe even unusable.

John: That's correct. And actually, his invention would be that the actual...it would stop immediately so that you wouldn't have a defective product, or at the very best, it might be something you can fix earlier.

Gene: And so, different way of working between the operator and the machine.

John: Exactly. And then, I think the bigger story here is that, something that I've read...where they...I've called this a generational culture, and it was that he understood some of these principles of making things more efficient but also understanding failure.

So, the melding of failure and efficiency. And this is something we talk a lot about in DevOps, about the counterintuitive nature of going faster and being more resilient. You can see this in the early looms, if you will. And,

turns out he makes money off the patents. So, he sells a lot of the patents for these really ingenious inventions, and then he turns around and he goes ahead and starts an automobile company. And his son gets involved, and that actually becomes the original Toyota.

So, I think when we talk about Toyota and that culture, it's really important to understand this father and how he thought, and we can see that in all the downstream discussions that we have about Toyota and Lean.

Gene: And so, that sets the stage for another extremely prominent figure, Taiichi Ohno, who is often credited for being the father of the Toyota production system.

John: I think that's where you have to understand the history of Toyota. Actually, Sakichi Toyoda's son, Kiichiro Toyoda, was tasked really with building this automobile business.

One of the constraints that they had that the American companies didn't was land. So, Kiichiro grabbed this genius young man, which is Ohno, to say, "Help me figure this problem out."

So, it actually became this early story of the early manifestation of what we today call "just in time." And so, Ohno...There was a lot of things that Ohno had done, but he became, really, the engineer that drove a lot of the efficiencies that we know about Toyota Production Systems, but also helped really drive this new way of driving flow and acceleration.

And again, I'll go back to that, the son inherited a lot of the properties, this culture and this brilliant engineer that just creates, really drives this whole movement, what we today call Toyota Production Systems.

Gene: You know, it's fascinating. Taiichi Ohno writes in his book that it's so much about constraints, as you mentioned, and he talks about this other constraint, which was just that market size was so much smaller. So, in 1949,

in Japan, the total market size for cars was one thousand cars. In this same period in the United States they were selling five million cars annually. So, it's so interesting that so much of this was born out of necessity, out of the destruction of post-World War II.

John, you mentioned that there is this culture that spans multiple generations. There's some amazing stories about how Toyota...they always felt that culture was their secret to success.

John: Yeah, one of the early stories with Kiichiro, when they were talking about a story about the original textile and loom company, that somebody had stolen the plans for a new version. I always like to put a literary license on it, but I can imagine everybody panicking and running up to Kiichiro and saying, "What are we going to do? They have our plans!" and him calmly saying, "Calm down, because by the time they figure out and put that in place, we'll be so much further along."

And it was just that kind of culture...that you couldn't cargo cult them.

Gene: In other words, their competitive advantage came not from the invention but from the organization creating the invention.

John: Yeah, and it was even...you know in a DevOps community we look at companies like Etsy. You can't just take their tools and put them in place. You have to understand why they think the way they do. And there's the famous Ohno story, very similar, where the...we talked about the American coming into study all the...what Toyota was doing.

Gene: This is in the 1980s?

John: That's correct. And they were letting all these people come in, and at one point somebody asked Ohno, "Why are you letting them copy...look at our stuff. They're our competitors." Again, I have to use my literary license, I

could see him almost snickering, saying, "They can copy our process, but they will not understand our culture."

Gene: You know, John, listening to all this, there's this fastening narrative here. You could say that Fredrick Winslow Taylor sets the stage for the dominance of US auto companies. Probably most visible in the figure of Alfred Sloan, the head of General Motors from the 1920s to the 1950s. This created a century of incredible and unprecedented prosperity, but it was all based on deterministic thinking. And then, Deming comes along and helps set the stage for Taiichi Ohno. And as we mentioned, Toyota sends an established economic and manufacturing dominance over those same US automakers.

John: Yeah, I think that this threat of...that you have Taylor is to Sloan as Deming is to Ohno.

Gene: I love it. John, let's talk about the elements of Lean. As per James Womack and Daniel Jones, there are really five key principles that underpin Lean. They are as follows: value, value stream, flow, pull, and kaizen. So, let's go through each one of them and talk about our favorite amlogs in the DevOps world.

So, the first is value. John, do you want to talk about that?

John: Yeah, and I think we've discussed this. In one of the earliest caricatures of the DevOps movement was that kind of aha to ka-ching. You know, how do you make money? How do you get from a sticky note on a board to value a customer paying for something, concept to cash. We've talked about Mark Schwartz's book *The Art of Business Value*,[10] that it is about understanding what business value means to your organization.

Gene: Yeah, I think in this day and age, to create value, whether for an internal customer or an external customer, increasingly it's reliant upon technology work. And so, and I think the big breakthrough in recent thinking is Eric Ries just showing that we know what value is. But the magic of the minimum viable product is: What is the cheapest way that we can confirm that there is actually some signal of demand? So before we spend a year building something, let's at least confirm that someone's even willing to give us an email address to get the service.

John: And if you remember our discussion about Eric Ries, he learned from Steven Gary Blank's *Four Steps to Epiphany*,[11] which was about the dotcom bomb where people would overproduce things. And what Eric Ries was saying is, you don't really know where you're gonna wind up. You don't know what your value is, so you had to minimum viable product, build-measure-learn, to get yourself to find what that value really is.

Gene: So, the second lead principle is value streams.

John: Yeah, so we've talked a little about the value stream mapping that comes right from Toyota, how Toyota really understood the flow of delivery, and in their case it was producing an automobile.

Gene: You know it's interesting that Lean started out in manufacturing, and then where the Lean Enterprise Institute went was to expand that to all sorts of different domains, whether it was payroll, to services, to healthcare. And so for years, they were spending...really focusing on how to expand the application Lean to all sorts of different domains. One of the things that Mike Rother did so well in his book *Learning to See* is really describe the exact steps to create a value stream. What are all the steps required to create value for the customer?

Taiichi Ohno created something called the Seven Deadly Sins of Manufacturing, and it was all around waste reduction. It's interesting, in the DevOps community we've actually changed the concept of waste and recast it in the concept of toil. There's this view that waste is dehumanizing. The goal is not to go on relentless waste hunts. It really is to reduce toil so that we can better create value more quickly and reliably.

So, the Sevens Sins, as Taiichi Ohno describes them [one], one, delay and the notion that anytime work is in queue we're not creating value. Obviously you want to reduce delay wherever possible. Second is overproduction, the idea that the worst thing that we can do is create parts or inventory that will never be used. Third is over processing. Now, I think in the technology world we would call this gold plating. Working on a feature that actually doesn't create value for the customer. Fourth is transportation. This is in the physical world and makes a lot of sense. Every time we have to pick up a part and move it, that requires time and energy, so we want to reduce that as much as possible. Five is unnecessary movement or motion. The theoretical ideal is really single piece flow, and in fact, there's something lovely about the assembly line because it never has to be transported. Six is inventory. Inventory inherently creates waste. The notion that as we increase inventory in the system, lead times go up. And then seven was the reduction of defects.

John: I was just going to say the inventory is interesting because that was really a big part of Goldratt and the Theory of Constraints of why it's an area where it really actually matches well. But I think what's really interesting to me is when we move forward and America's notion of what Toyota did, of what Ohno was creating. We had all the Lean manufacturing discussions, but Mary Poppendieck, she wrote this seminal book, in my opinion, called *Lean Software Development*.[12] And I think it was in 2003, and I think the brilliance of this book is that she took basically the Lean principles of

manufacturing and translat[ed] them into the software development world. Like you had pointed out, there's certain things like transportation or things that don't translate as well. But what she did, because she had a strong, Agile background—actually, her husband, Tom Poppendieck, as well. They work together. Lovely couple, if you will. But they defined their version of the Seven Deadly Sins of Software Waste, and it matches sort of well, but again it was really in the construct of "How do you deliver software?" The first one is partially done work; that's reasonably easy to understand.

Gene: I think that one's a great one, just because I think in some ways that's the stage for DevOps—that code that was completed but not in production is a form of waste.

John: Yeah, absolutely, and then a second one, described as extra-processed. But if you go through the book you actually see it really is amplified learning. It's decide as late as possible, that is our MVP/BML concept—deliver as fast as possible. And power of the team. This is a collaboration. You can see how even their transition, with the Lean influence on us in DevOps, but what they did is really taking these waste principles into delivery of software.

Gene: I think even there we see this notion of nondeterministic thinking. The idea of preserving optionality, making decisions as late as possible. I think those are all something that was very novel in the software space.

John: Absolutely, and even part of this extra process waste, which is seeing the whole system-thinking. And then, just to follow through the last four of their waste, which is extra features. We see that all the time. Again, that's an anti-pattern of whatever Ries talks about. And then, task switching.

Gene: Task switching is so interesting because it was very disruptive in the manufacturing space, but it is orders of magnitude more disruptive in knowledge work or technology work. Because the cost and effort required to deal with cognitive interrupts is so much higher with this example of just being able to sequence a bunch of shapes. The fact that if you're being interrupted it took ten times longer. So, something as simple as sorting shapes—think about something as complex as actually working on something genuinely hard.

John: There have been points in my career where I've done software development, and sometimes when you're building—I've had a couple of products that I've worked on for a year, my own software product. And sometimes you are three, four hours deep in the architecture in some error routine, and then somebody calls you out for a meeting. It literally sometimes takes, for me, almost an hour to get back to that place. So that really is, for knowledge work, an incredible form of waste. And then just to finish the last two that they defined, which was waiting. And I think that was a common theme between both manufacturing and between Lean software development. And then defects, of course.

Gene: The third area is flow. I think this might be very abstract and maybe nebulous, but the Lean Enterprise Institute had a very great definition for this specifically: "Make the value-creating steps occur in a tight sequence so that the product or service will flow smoothly towards the customer."[13] I love that definition. I think it may be odd to put this in this section, but I think the best treatment of flow really comes from the Theory of Constraints. That in any flow of work, there's always a dependency on one bottleneck. We talked about how any time lost at the bottleneck is time lost to the entire system.

And Dr. Steven Spear in his great book *High-Velocity* [*Edge*], he talks about two specific capabilities that every organization needs. They need to see problems as they occur and they need to swarm them. In other words, when something goes wrong, we elevate it to potentially the entire organization because nothing's more important than to restore service, especially at a bottleneck. And I think the theoretical ideal of how to maximize flow is called single piece flow, sometimes known as one-by-one flow. That means batch size of one, buffer levels of one.

My favorite example of what can go wrong and what also can happen in terms of miraculous recoveries is in [the] 1997 Toyota brake fire. This is famous because there was a fire at a factory in Japan that was the single source of what they called the P valve. It's a ten-dollar brake part, but the fact that there was only one source essentially brought down the entire fourteen-thousand-vehicle-per-day production capacity. People were predicting that it would take weeks for Toyota to recover, and the comments were even predicting that for every day this factory was offline, Japanese GDP would be reduced by .1%. What actually happened is remarkable. The *Wall Street Journal* wrote one week later that Toyota had resumed 90% of its normal output,[14] less than a week after the fire, and what's amazing is how they did that.

Essentially Ason, the supplier, shared all their design plans with every other manufacturer in the supplier network. They were trying to figure out who could potentially create capacity to build this part. They were grabbing equipment from all over Japan, and Toyota even dispatched nearly a thousand engineers throughout the supply chain to help solve and restore capacity. What I find so amazing about the story is that in most organizations they've used a constraint within their buildings, within their production lines, but in the case of Toyota, this spans the entire Toyota

supply network. When there was a problem at Ason, everybody knew that it was in everyone's best interest to help Ason get back on their feet.

John: You know, that's a great story. We talk a lot about flow in general, and we've talked about Eric Ries, and before he wrote *The Lean Startup*, he wrote his S*tartup Lessons Learned*,[15] which was basically his blogging story of what he did at this company called IMVU, and I think there's a great flow story there. When you get to databases, that becomes the brick wall, if you will, for change, because you have scheme changes. It's very hard to match change control, scheme change, the impact of the change. They actually created a model where they actually just created new records so they actually were able to have multiple copies of records and know which was the current record. It was just a great example of how they put everything together about speed and efficiency and flow.

Gene: This is so...I mean, I think to talk about the risk, the database scheme change, is always so scary because often you only find out about coding errors at runtime when lookups and queries break. What they did is so amazing because they said it is actually better to never change a scheme, to never delete data. Well, actually change of code to ameliorate those issues to optimize for flow as opposed to data purity.

John: Now, I love that story, Gene. I think another story that fits here is if we go back to Eric Ries. We know him from his *Lean Startup*, but before that he had his [*Startup*] *Lessons Learned* blog, which were really just stories of how they implemented efficiencies at this startup called IMVU. The one story I really love about that is how they dealt with database changes. So, scheme changes...that's typically when we talk about delivering service and software, that's the one place where we get into.

Gene: It's terrifying. [*laughs*]

John: Right. That's the part where we have to worry about change control, because we really do have to worry about...What they did is they came up with a model of never really deleting or editing a record. They basically would only add a new record and leave the other one, and they would just flag what was the additional record. The idea was just...to a database administrator, it just seemed so foreign, but from an efficiency and flow, it was thinking outside of the box.

Gene: All right, this is another great example of all these continuous delivery practices that were created at IMVU, and when I read that one, I frankly thought that this was the most idiotic idea ever. Why would you keep multiple records and copies in a database? And yet, their decision, I think it's absolutely brilliant, is that to optimize for flow, we can keep multiple copies and instances of data, and for a whole bunch of reasons it's safer and faster and great for flow. And of course, John, we know that we can't have any discussion of flow and DevOps without bringing up Dr. Donald Reinertsen and his amazing book *The Principles of Software Development Flow*.[16]

He extends some of the ideas that were first introduced in the Poppendieck book and takes it to a level that is impressive and sometimes intimidating. He talks so much about the notion of what it takes to improve flow throughout the entire software development process, and again that idea of preserving optionality, making decisions as late as possible. Just a little aha moment for me. I think the learning for me was how operations and QA can best enable flow is making sure that there's fast feedback loops for developers. In other words, we want to make sure that if there's an error, we want the developer to ideally learn about that within minutes, worst case hours, as opposed to having to wait nine months later and then only learn

during integration testing. We have to give that feedback while the link between cause and effect are obvious and strong.

John: Yeah, and I think that one of the things there too, if you look at the history of DevOps, there was that this...they had originally, when we were talking about moving the fast flow for delivery of software, the QA people were concerned that they were going to be left out or "How are we going to get into it?" What we did is we invited them in, that collaboration where we put their knowledge into the automation of the flow. Again, another example of thinking about flow by pulling groups into the project as opposed to throwing it over the wall and being a separate process.

Gene: And another great example [of] how we're taking the expertise that is traditionally in people's heads and putting them into the automation. The fourth area according to Lean is pull. The Lean Enterprise Institute describes pull as the ability to make it easier to deliver products as needed, as in just-in-time manufacturing or delivery. With increased due date performance, the customer can pull the products from us as needed. So, this allows us to build to order instead of having to overproduce sometimes very expensive inventory. John, the kanban card is probably the most easily recognized tool in the Lean arsenal, often used to signal replenishment. I learned from you that this didn't actually come from Toyota.

John: Yeah, it's interesting. You have Taiichi Ohno, and he basically, in his quest as we talked about earlier for *Just In Time*, his constraint again is he doesn't have the land that the US companies have. So, there's this story where he goes over to visit General Motors and Ford to try to help him understand some of the efficiencies that they get and...turns out that he really doesn't get to solve his problem there. So, he's driving somewhere, and I guess they stop at a grocery store, and this grocery store, it's actually called

Piggly Wiggly. It's big in the South, but I think this was in Ohio. As he's in the store, he sees how the shelves and the backlog, if you're looking at cereal cartons or milk cartons, there's a shelf and a list, and he notices that...Maybe he got lucky, and he saw a stock person. You get down to two and somebody added the three back so that there was always five. The story was that was the genesis of his aha moment of how he could solve his "just in time" problem.

Gene: So, specifically, I can imagine on a grocery shelf at Piggly Wiggly there would be five cartons of milk, and if someone takes one off the shelf, it would almost immediately be replenished from the back.

John: Yeah, I think in the grocery store model there was somebody who walked around and made sure they were full, but in his world, he thought about that in terms of how to improve that in parts, and you're exactly right. And in kanban from hardware, there was a signaling system: part goes, signal replenish.

Gene: That's awesome. So, if you're in the manufacturing plant and you use a part, you would immediately take this card, which would immediately signal replenishment of a part, batch size of one.

John: Again, just the irony of he goes to Ford and General Motors to get his answer, but his answer comes from a silly named grocery store, Piggly Wiggly.

Gene: The notion of kanban in our world, I think most of us were introduced to it by a famous book that David J. Anderson wrote in 2010 called *Kanban*.[17] What's amazing is that this book really codified how he improved the flow of work both at Microsoft and then later at Corbis, the stock photo company. What's interesting to me is that I first ran into his

work in the Theory of Constraints community. In fact, when I was taking this graduate course at Washington State University. In fact, I think he won the paper of the year award in 2005, and that was based on his work at Microsoft. But what he did after he wrote that paper was come to this conclusion that all the work required to apply Theory of Constraints was maybe unnecessary.

He found it very burdensome, and the shortcut, the heuristic he came up with, was essentially the kanban board, the notion of creating one column for each work center and then putting strict WIP limits in, work in process limits, just like at Piggly Wiggly. A developer can only work on one thing at a time. When they move work from left to right to the next work center, only then would you pull in work from the backlog.

John: Yeah, no, I think one of the stories that you read...David Chanison wrote a blog about his epiphany for kanban for software. Which was, he said he was on a plane ride and he read *The Goal*. I think everybody knows what the kanban board is. To me, I always like to think the kanban for software is "to do," "doing," "done," and you're always pulling. You have these, what we call "WIP limits" inside, or only three things could [be] in one column, or the number [of] thing's in a column, but the kanban for software and again, we talked earlier about value stream mapping being the critical tool. Kanban for software is equally critical in high-performing organizations and how they visualize because that was the important thing, it was the visualization of how we conceive flow in services software delivery.

Gene: You know, in *The DevOps Handbook* we had a lot of case studies. Some of my favorite case studies were the ones where it was by pure application of kanban. Without automation, without environments of ill on demand, without automated testing, it was these amazing breakthroughs in

throughput and flow were created just by limiting worker process and visualizing work. I thought that was just astonishing.

John: Yeah, you know another good example is Jody Mulkey, he's the CTO over at Ticketmaster, another good friend of the DevOps Enterprise Summit. I got to visit their office, and literally, you can't go more than about thirty or forty yards without seeing a kanban board on every floor. So, the idea that he can actually walk around as a CTO and not even ask the people, he can actually see the visualization of all the departments working together to create the value in the Ticketmaster organization.

Gene: The last area is kaizen and the pursuit of perfection. My personal favorite description of this comes from Dr. Steven Spear. He says, "There is an internal pressure for high velocity discovery, learning, innovation, and invention. This is important because we cannot design for perfection, because reality is too complex. Therefore, we must pursue perfection."[†] I love the observation that Spear made that said when studying high performers, we are often dazzled by their altitude, but instead of being dazzled by their altitude, we should instead focus on their rate of climb. I love that because it shows that what high performers do is that they know that in order to win in the market place requires them to out-learn the competition, and that's why the best are always getting better. John, you have another favorite description of kaizen and the pursuit of perfection.

John: Now, Gene, we've talked a lot about these two books, Mike Rother's *Toyota Kata* and Steven Spear's *High-Velocity Edge*. And, in fact, a lot of times I tell people, "You should read both of these books in order of *Kata* and then Spear." And one of my favorite parts of Rother's book *Toyota Kata*, he describes something called "improvement kata," and improvement kata is a four-step process. And the first step is this challenge, or he describes it in

the book as a "true north." And that's your perfection, it's this ideal, this thing in the ideal. And so you literally start with your current condition, step one, or actually step one is the perfection, step two is the current condition, you work your way to the target condition, and then your PDCA—plan-do-check-act—your way in these steps, that climb, if you will, that you described.

And we recently got to hang out with Mike Rother in Detroit, and he does this thing now, what he's calling, "kata in the classroom." And it's really cool—he takes puzzles and he puts teams together and he makes you put this puzzle together in this interprocess because he uses the four-step process where he says, "I want you to create the puzzle and put it together in fifteen seconds." You basically have to go through step one, step two, step three, and it turns out, I went up to him and asked him, "Has anyone ever done this in fifteen seconds?" And the answer is nobody out of all the times he's done it.

So, it's a great example of this ideal perfection, and I think it's the best way to describe when we've talked about determinism versus non-determinism. In a deterministic way, you would say, "Fifteen seconds. If you don't get it done, you fail," right? In a nondeterministic view, it allows you to work towards that perfection without a failure. In fact, the success rate of getting closer and climbing, if you will, is an example. And that's why I really love his improvement kata.

Gene: That's so great. In fact, it reminds me of another theme that showed up in the business literature. In Jim Collins's book *Good to Great*,[19] he calls it the big, hairy, audacious goal, the BHAG. And he even says, the pursuit of the unattainable is what enables greatness. And so the whole notion of the unattainable perfection, zero defects, one-by-one flow. It is only through the pursuit of that, that we can create greatness.

You know, it's funny that you bring up Mike Rother, because I took his original Toyota Kata workshop in 2011. And one of the introductions to talking about his material was talking about the Andon cord. And so, I think almost everybody knows these days plants modeled after the Toyota Production System. On top of every work center, there's a cord that everyone's trained to pull when something goes wrong. So sometimes that means, "I made a defective part. I got a defective part from somebody else. I have no parts to work on." And even if an operation takes longer than documented, the documentation says it's supposed to take fifty-five seconds, but it takes a minute twenty, the action really does specify it as you pull the cord, and everybody knows now that when you pull the cord the entire assembly lines stops.

So, I knew that going into the workshop, but what I didn't know was how many times in a typical day the Andon cord is pulled at a Toyota plant. And the surprising answer is 3,500 times a day. And I think that's so interesting, because it's so contrary to how most of us have been trained. I was thinking about the way I was trained as a first-line manager was; my job was to buffer errors. You know, solve my problem locally before it caused a global disturbance, and it seems like the Andon cord is the exact opposite. We are amplifying a small problem and elevating it and potentially disrupting the entire organization. And I think this goes back to Steven Spear. He said: This is a great example of how we have to see problems that occur and then swarm them. Not just to solve problems quickly, but to create new knowledge. And that is a part of relentless pursuit of perfection.

John: You know, there's two really good Andon cord pull stories, and one is in *Toyota Kata*. One of the things that Rother says in the *Toyota Kata* book is, which is fascinating to me, is that when you pulled the Andon cord, the first thing the floor manager would do is come over and thank you. And the reason was because you were thanking that person for creating a learning

opportunity. And one of my favorite stories in that book also, I believe it was a plant in Tokyo or Tokyo City or Toyota City, where they would average a thousand Andon pulls a shift. And for some reason, they went down to eight hundred. And in kind of a Western culture or a Taylor-Sloan model, that might have been "Yay! Two hundred less defects," right? But no, at Toyota, the plant manager got everyone in a room and said, "We've got a problem here." And the problem was that they were learning two hundred times less a shift.

And there's actually an even better story about the Kentucky Toyota plant. Which was where Toyota did their first...really on their own plant. And I think the story was, they were producing well over two thousand cars a day. And it was either an analyst or an auto industry analyst who asked the question, "How do you pull two thousand cord..." I mean, "How do you go ahead and create two thousand cars a day?" And the answer was, "Oh, that's easy, we pull the Andon cord five thousand times a day." Right? So, if you can understand that logic, that that type of resilience creates that speed.

Gene: Yeah, to add onto that amazing story. It's so astonishing what happens when the Andon cord pulls goes down. Often what they will do is tighten up the tolerances to increase the frequency of those Andon cord pulls. And that reminds me...Sometimes it's tightening up the tolerances for quality, sometimes it's reducing the buffers so that it actually creates more disturbances so that's what allows the...We just increase the value of what perfection is. I think in the DevOps world, I love how this manifests itself into the work we do in sometimes surprising ways. I mean, some of the behaviors are, I think, very commonplace. The whole notion that when someone breaks a build or breaks an automated test we don't have to stop everyone from working, but we certainly would expect that engineer who broke the build or broke the test to stop whatever they think they should be doing and work on how do we restore it back into a green state.

But I love how these sort of behaviors also go onto other domains of work. The notion of peer reviews. Instead of relying on chained approval boards, we rely more on getting fellow engineers to review our work. And so, if I'm doing whatever I think is the most important thing, and someone asks me, "Gene, can you take a look at my code so I can push to production?" Really, I should drop whatever I'm doing and help that person get into production. Because we know that the longer that person has to wait, the worse the outcomes. And so, tomorrow it might be the other way around. I might be the person who needs to get into production, and I need them to drop whatever they're doing. But I think you can see it in another place, which is the blameless postmortem. You know, the classic ritual we have is that once something goes wrong, we can create the conditions so that someone can safely tell stories about what happened, capture the chronology, and more importantly, the assumptions that we had that led to the accident.

But, what's interesting to me is that Randy Shoup put out, for many years he was the App Engineering Director, and he said—

John: Google.

Gene: Yeah, at Google, thank you. He said: We found that, because we kept having these blameless postmortems, we didn't have enough customer-impacting outages to have postmortems on. And so they said, "We did them not only for customer-impacting incidents but team-impacting incidents." And it's just this recognition that if you don't have enough learning opportunities, you cease to get better.

John: Well then, speaking of Google and tying it back to that Andon cord, about three or four years ago, there was a presentation where they showed that they did 75 million automated tests a day, and then last year there's been

an updated version of that, it's been 150 million automated tests a day. And what I say to people is, that's that Kentucky plant, and the Toyota Kentucky plant is when you've got that built-in resilience. I mean, that's their Andon cord—150 million times a day they're stopping and testing stuff and catching breaks, things that would otherwise break. And you know, the last thing I would say, somebody said, "When's the last time that you called support for a Google product?" You don't.

Gene: John, this is great. We talked about how Toyota went on to dominate and decimate the US auto industry. What happened as a response as the research committee was mobilized to figure out how to econify what is Toyota doing differently and how can that be adopted in other organizations to regain parity or even supremacy? We talked about the five classic areas that Lean defines, in terms of the practices that we call Lean. As really, we have to acknowledge that Lean really is how most of us get introduced to this entire domain.

John: Absolutely, Gene. You know, if we look at it from the beginning of the DevOps movement, we've learned so much that a lot of what we do is from Lean. Lean has been the driving force; it was in *The DevOps Handbook*. What I get excited about too is if we look at Lean as this thing that we can sculpt around, next in the future modules here, we're gonna look at how we can take safety culture and the resilience engineer and the works from learning organizations on top of this body of work for Lean.

[†] The full quote is: "There is also internal pressure for high-velocity discovery, learning, innovation, and invention. The work of organizations is incredibly complex, and nothing complex can be designed perfectly. There are simply too many parts—be they the actual components of complex technical systems or the contributions of people who are expert in myriad disciplines—

connected in too many convoluted, interdependent ways for a small group of smart people to plan a system that will work adequately—let alone perfectly. Since you cannot plan perfection, then you must pursue perfection."[18]

Module 5

Safety Culture

Gene Kim: John, you've long asserted that there are two major influences that DevOps draws upon the most. The first was Lean, which we've already covered. But the second is without a doubt safety culture. So before we deeply go into safety culture, let's first talk about how it got into DevOps in the first place.

John Willis: Yeah, Gene, in my role as the DevOps historian, I think you go no further than look at John Allspaw. And we've seen John Allspaw and his influence in what he did at Flickr in his presentation. But here again, over a ten-year period, John introduces a whole nother field, this resilience, safety, human factors. And not only does he introduce these authors and these works, but he helps translate how they have impact and, in the end, helps us understand how to use this knowledge in DevOps.

Gene: Holy cow, it's John Allspaw again, and maybe it's no surprise that he shows up in the center of so many of these pivotal moments. So, safety culture is like Lean; it's such a large category that incorporates so many

subcategories of slightly different thinking. How would you summarize safety culture, or as they have seemed to rename themselves, Resilience Engineering.

John: Yeah Gene, as you described, it's a large body of work. So, summarizing one simple section would be a little difficult. I think if we look at it, it's a combination of a number of things, as you describe.

Some people would call it human factors and how we think about the human factor of how we think about complex systems. It is systems thinking. It's dealing [with] complexity. It's, literally, safety is another name for it. And then Resilience Engineering is now something that is more prominent.

We've recently had an opportunity to meet and talk to Dr. Sidney Dekker, one of the leading authors in the space, and we cornered him and we asked him, "Okay, what is this thing?" And he said, "You know, we can just call it safety culture." And he said that a simple definition is, "It's a culture that allows people to give the boss bad news." That simple. For me, I think, there is an underlying narrative as you read the works of a lot of people in this space. You see clearly that this idea of...they make a differentiation they called Safety-I and Safety-II.

And what's interesting in the way I look at that is when we go back to when we talked about Taylor and Taylorism and how that was a model that was seemingly deterministic in terms of the way things flowed, X caused Y. I think you can clearly look at Safety-I and see that in that same pattern. And of course, all of these authors that we are going to address and talk about are all focused on Safety-II. Interesting enough, you can compare Safety-II to really a kind of Deming and nondeterministic thinking, where you're dealing with complexity, where you're looking at the systems approach

of...How do you deal with complexity? How do you deal with systems in our world? How do we make this work with this DevOps?

Gene: And I love the way that Dr. Dekker describes it. He says that Safety-I...that the question we ask is always, "Who caused the problem?" Whereas now in Safety-II we ask, "What caused the problem?" And so in the world of Safety-I, we have a tradition of identifying the person...you know, find the person who made the error and then maybe the right thing to do is get rid of that person. Problem solved. Whereas [what] Safety-II seems ingrained in is the understanding that world is far more complex than that, and by making the person the problem we deprive ourselves of the ability to learn from mistakes and actually really prevent it from happening again.

John: You know, along those lines, you told me a really good story once that you heard from Dr. Dekker, Dr. Sidney Dekker, about the B17 bomber.

Gene: Yeah, I thought the story that he told at DevOps Days Brisbane in 2014, there was a specific story that really helped me understand the field better, and it was about the B17 bomber.

So, the B17 bomber in World War II was the first plane to have hydraulics powerful enough to power both the flaps and the landing gear. But in 1943 they had this problem, they would come in for a landing, they would safely land, and then the pilot would essentially crash the plane. And so, they determined that what the pilot was doing was that as they were landing, they would lower the flaps, they would lower the landing gear, they would land, they would raise the flaps, and then the pilots would accidentally raise the landing gear. And so, this would not only cause the plane to crash, but it would actually cause the propellers to hit the ground and therefore damage the engines.

So, they were looking at how to solve this problem. And they found everything they tried didn't work. They tried checklists. They tried training. Dekker tells the story about how they even looked at the pilots as a whole, trying to figure out which pilots were more prone to make the error and take them out of the rotation.

And as the story goes, he says that there was a mechanical engineer who was in a cockpit who noticed that the controls for the flaps and the landing gear were identical, and they were right next to each other. So, he hypothesized that maybe the pilots were accidentally in the act of raising the flaps, also raising the landing gear.

And so, he came up with a clever countermeasure, to make a little wheel which he applied to the landing gear controls, and a little pair of flaps and attached that to the flaps control. And as Dekker tells the story, the problem never happened again.

So, when I heard that story for the first time, I think I finally understood why they like to call this field human factors. In my mind, it's really like the modern version of user interaction design, or UX, as probably best known in our world by Alan Cooper, who wrote the famous book *Inmates Running the Asylum*,[1] about how we, as software engineers, often write very inhumane, mean systems that make our users cry.

John: I love that story as well. And I was actually at that event. See, he was day one keynote, and I was day two. And I had read some of his work, but I had never met him. So, after the presentation, of course, I went up to him. "I have to talk to you." And at the time I was doing a lot of research at Deming in nondeterministic thinking. I said, "It sounds to me your narrative of everything you talked about, kind of the history for safety, is a nondeterministic thread." And I remember he said, "Oh, my God, I haven't thought about it that way." And I had actually started a journey of him being

introduced more to these concepts that bleed a little bit into Lean. But anyway, it was just kind of fun that I was there, and I loved that story.

Gene: John, we'll be talking more about the history of the field. But I think that we both agree that Dr. Sidney Dekker, if he's not the clearest communicator, he's definitely one of the funniest communicators who made these ideas accessible to the rest of us, including John Allspaw. I think the B17 story is so great. Safety-I shows all the things that didn't work. You blame the pilot, you train them, you give them checklists, you punish the pilots for nonconformance. Where Safety-II is what actually made the airplane safer because they made the airplane safer to fly.

John: I totally agree. I love that, Gene. I think with that, that it's important then for us to look at the history of this whole area of work. And you had mentioned to me that you had actually some experience over the years with some of this work.

Gene: Yeah, I am laughing because it's an embarrassing story. So in 2005, George Spafford, one of my *Phoenix Project* co-authors, we were co-authors before then in a book called *The Visible Ops* [*Handbook*].[2] He actually introduced me to this body of work.

We read *Normal Accidents*[3] together by Dr. Charles Perrow, and that was all about the Three Mile Island disaster, about high-reliability organizations, especially around aircraft carrier flight operations. And we read this book called *Just Culture*[4] by Dr. Sidney Dekker, and I'm embarrassed that I read all of these books. And I sort of nodded and said it sounds good and really missed the whole point of the books. I didn't make the profound connections that you have been pointing out until you and I talked three years ago. And I just wanted to put that in contrast as, where I had just read it and put it on my bookshelf, John Allspaw reads it for the first time and

writes this seminal blog post that helps define the field. So, there's difference between reading and fully comprehending.

John: Well, and I think that was so important where he...the title of his paper was "How Complex Systems Fail: Web Operations Perspective."[5] That was the transition and the leap; otherwise, it was just a whole bunch [of] work out there, so yes.

Gene: John, even given my caveman understanding of the field in 2005, I did know that the name that kept on showing up over and over in these books was Dr. Erik Hollnagel. So, tell us about Dr. Hollnagel's contribution to safety culture.

John: Yeah, I think as you go back and you read all the works, you somehow wind up finding his name involved in it. And I think one of the things for me that stands out, how he got on my radar, is what he coined as this efficiency thoroughness trade-off—ETTO. And he really was this balance, these systems. You're always going to have these tensions between efficiency and low investment sufficient to achieve the goal. And thoroughness, which is about the completeness, and to me that resonates in that speed and resilience discussion. And what he was telling us is that there's always going to be a balance between the two. I thought that was interesting, but I think you had mentioned that you'd thought about this from some of the Goldratt work in the core chronic conflict.

Gene: Yeah, in fact, one can't help notice that there's a similarity between the efficiency thoroughness trade-off. Right, so given the Hollnagel language, do you strive to get sufficiency through efficiency, or do you want thoroughness, where you want to achieve the objective without wanted side effects? Doesn't that sound familiar? The Goldratt notion of core chronic

conflict. You want to get to the market quickly, and that means we need to make changes quickly. Or do we want to ensure reliability and stability and security, which means we need to make changes carefully and deliberately at the same time? I think where it gets a little divergent is that I think Hollnagel would say that it's always a trade-off, whereas Goldratt would assert that there's always a way to get both. Or we strive to find ways that we can break that core chronic conflict.

John: Yeah, and I think...Not to say that we're absolute experts in this work, and we're just hopefully introducing these works to other people, but I think the way I read it, there was an equilibrium. If you read it at face value, it is a trade-off. And it is called a trade-off, but it's the balance of the trade-off and to what degree. And we talk about speed and resilience. Then in DevOps, we've got a lot of evidence that we can get both.

But I think that if you look at the Google SRE story about something called error budgeting...And I'll talk a little about that in a second...But when you get to Google's scale, there's possibly a point where you actually, what Hollnagel was saying, is there is this final boundary of truth. And so one of the ways that Google dealt with this kind of truth at scale was they came up with this thing called error budgeting. The idea was, each kind of service, or service owner, would create their own service-level objective, and commonly people would think about these as SLA, service-level agreements, and so you would define these things into percentages of uptime and downtime for servers. If I said three nines, it means that the expectation was for the service [to] be up 99.9% of the time. And that's common, three nines or four nines.

So what Google did with error budgeting is they said, "Okay, let's make an agreement of what the SLO is for this service, and if it's three nines, 99.9%, then we'll say that you as a development or application or team can basically go as fast as you want, as much you want, until you hit the .1. You

know, the minus of the 99.9%. And then we're going to stop you for that month. I thought that was brilliant, like rubber heats the road. You can't just always, at some scale, go faster and be more resilient. There is a reality check, and I think that Google is the perfect way to show us that balance.

Gene: I love that as well, because in my mind the error budget construct is one of the best examples of a self-balancing system. If the developers want to go fast, it can go as fast until they hit the brick wall where they've lost the right to go fast. On the other hand, it rewards ones who can do things well, and they can go as fast as they can demonstrate where they haven't lost control defined by the service-level objective.

John: Yeah, and if I'm correct in reading Hollnagel's definition, it really is in equilibrium. It's not an either/or.

Gene: What I also love about the error budgeting model, it shows that even Google has to make trade-offs. We all take shortcuts, and sometimes there are real consequences for the decisions we make. If we can't deploy anymore, what it forces us to do is pay down technical debt, and in the ideal, the outcome is that we get to go faster again while preserving our reliability objectives.

All right, so we just covered Dr. Erik Hollnagel. Without a doubt, another giant in this field has to be Dr. David Woods. So how does he enter the picture?

John: I think as we look at a lot of these authors...We've talked about John Allspaw, and his introductions, or Dr. Sidney Dekker, or Dr. Richard Cook...If you talk to them, which we have, they'll all point back to Dr. Woods today being the leading researcher and really the person who's really extending this body of work. And I think that's important.

If you look at Dr. Woods today, a professor of cognitive systems engineering, human systems integration at Ohio State University...But the thing that I think makes Dr. Woods interesting is he has taken his work in a lot of different fields. He's looked at critical care medicine, aviation, space missions, intelligence, government intelligence...just across the board. So, he keeps validating this information. He also, as we both know, was basically a part of both the Columbia and Challenger action investigations for NASA. So, he's worked with NASA on those works.

One of his early works is something called *Behind Human Error*,[6] which the original version was in 1994. Dr. Sidney Dekker was a student of his. What's interesting here is that I've tried to go read Dr. Woods, and when I was reading, I was like, "Oh, my God, I don't understand this." But then when I talked to [Sidney] Dekker and [Richard] Cook, they were like, "We don't either."

I think you point out something interesting about Dr. Dekker. He's very good at translating some of the core principles that you get from Dr. Woods. The thing that really got on my radar of why I understood that Dr. Woods was important to our field was, here again, something that John Allspaw introduced to us with his master's thesis. It was actually based on another old 1995 work by Woods, basically where he addresses dynamic fault management and abductive reasoning. When these complex things are happening, how do you deal with that? Abductive reasoning, all that complexity. And if you look at the characteristics, just to summarize the work quickly, basically what he says, and again, John was able to translate this into his work at Etsy, which is a brilliant...You look at John's master's thesis,[7] you see all of this, which I highly recommend.

When people are trying to deal with these anomalies, or dynamic fault incidents, they're using their best mental model of a very complex system. So, they're best intentions, but they are just these abstractions. The way of

which we get our feedback, the anomalies, the monitoring, by definition are latent. So, in a complex system, they're moving so fast, I'm seeing a signal about this, I mental model around this...

There's also this idea of safeing. In a complex failure of fault management, there are some things you just have to fix. You have no choice but to fix this. Even though it might disrupt your ability to understand how to further fix the problem. And last but not least, the complexity, the latency, there's issues that create other issues, that some issues show up further down the line...These are just complex systems.

All in all, what Woods and then Allspaw beautifully summarizes in his thesis is you have to have a new way of thinking about these systems. I would say that it's a nondeterministic way. You can't just say deterministically, "We saw this last time. You hit this button. It must be fixed." They're trying to tell you that in these very complex systems, you really have to think much broader. You have to use systems thinking. You have to embrace complexity.

Gene: I just have to say, I, too, found Dr. Woods's work often to be difficult to understand, and I love that. Just like Dr. Sidney Dekker helped make the work more accessible, here again we find John Allspaw doing it, but not in the abstract, doing it in the extremely concrete. Showing how Dr. Woods's very important work, how it can directly impact and improve the work that we do in highly complex systems.

John: John mentions that he hates when people look like fanboys, but I can't emphasize enough how important it was for him to do a master's thesis as a part time...to take this incredible body of work and really apply it to actual large-scale outages at Etsy at web-scale. Nobody had ever done that before.

Gene: And he completed his master's thesis in 20—

John: I think it was ’15.

Gene: In 2015.

So let’s talk about two types of concrete patterns that we see in DevOps that stem from this work. The first are the technical and architectural practices, and second is about the softer side of culture. Let’s talk about the technical and architectural practices first.

John: Yeah, I think to do that we have to or should go to Dr. Richard Cook and how he explains to us complexity. But, Dr. Cook is a colleague of Dr. Woods. He’s worked with him over the years. He’s actually a true doctor. Which is kind of a joke. He actually is an anesthesiologist, and interestingly, he seems to practice and does patient care, like safety resilience, then he practices. The other thing about Dr. Cook, he was actually, again through Allspaw, introduced to Velocity Conference.

But he’s been very active in engaging in our community. He’s been speaking at DevOps Days. He’s constantly on Twitter in the DevOps conversation. In fact, it’s very easy to make a comment...I’ve had this happen to me a few times...I’ll make a comment about something, he’ll question it, and we’ll actually jump on Google Hangouts and have an hour conversation. And others have told me, “Oh, yeah, that’s a common thread.”

So, I really appreciate that he’s come into our community and probably shown the most...John Allspaw was already in it...Dr. Cook...We have to look at, you know...We talk about [how] John Allspaw introduced, in 2009, this blog article of how complex systems fail web operations. It really was just, by permission, a reproduction of Dr. Cook’s eighteen points that he had made. I think if we just look at these points, you know...

Number one, complex systems are intrinsically hazardous systems. Number two, complex systems are heavily and successfully defended against failure. We build moats around these complex systems. We, over the years,

try to defend those, and those become added complexity, and now we have to deal with them. Number three, catastrophe requires multiple failures. Single-point failures are not enough.

Gene: I love that language. Essentially what he's saying is that it's never just one thing. It's ten things that all go wrong at once, and that's what causes the fury crash.

John: Number four, complex systems contain changing mixtures of failures latent within them. Number five, complex systems run in degraded mode.

Gene: John, I love this one because it's this observation that not all failures result in a fury crash. Sometimes only after the fury crash we discover that the system had been limping along in a severely degraded mode due to great engineering, maybe for hours or even days.

John: Number six, catastrophe is always just around the corner. You know, we've gotten to know Dr. Cook, and you can hear his soft but strong voice saying that to you. "It's just around the corner, John." Number seven, post-accident attribution to a root cause is fundamentally wrong. Let me say that again: post-accident attribution to a root cause is fundamentally wrong.

Gene: John, you finally convinced me that it is safe to say that there is no root cause. What was the argument that you had made that finally changed my mind?

John: Yeah, I think that most people probably listening now, they would probably be, "What do you mean there's no root cause? I've been doing this my whole..." That's the argument I always get. "I've been doing this forever, John. How can you tell me there's no root cause?"

What I think I've learned, and what I think I've helped you understand, is we're saying, why should we stop learning? If we stop at a certain point, we've not enabled ourselves to really explore systems thinking. There could be lots of other things that could be part of this. So, it really is more of a kind of learning journey, where you're not stopping, "Yep, we got it. We're done."

Gene: There's a very decisive argument that said, "What caused the root cause? And what caused that?" Eventually you always get to the big bang.

John: Number eight, hindsight biases/post-accident assessments of human performance. Cognate biases. Understanding the role of cognate biases in what we do. Again, Richard Cook, Dr. Cook, was talking about this in patient care. But as we think about the mistakes we make, the different cognate biases, particularly hindsight bias, will always be part of our narrative. We need to understand that in complex systems.

Nine, human operators have dual roles as producers and as defenders against failure. This is a great developer story in DevOps. The developers are the producers. We decided early on at DevOps to give them pagers. We've talked about this. Now they're the defenders when the pager goes off. All practitioner actions are gambles.

Gene: Anyone doing our type of work, we all know the feeling when we type something in command line and then immediately think, *Uh-oh, that shouldn't have happened*. I love that phrase. All practitioner actions are gambles.

John: Number eleven, actions at the sharp end resolve all ambiguity. All organizations have levels of ambiguity at the edge, where it is met, and then bias follows. I think I've mentioned the story that one of the early outages at

Amazon, where they pretty much lost...everything was down. And they literally couldn't figure out what was going wrong.

One of the safeing, if you will, or what they tried to do, which was they decided to throttle the API. If you notice the early days of Amazon, everything was API. They sort of, actually it really is today, but...And so, by doing that, that was the sharp edge, and it created a whole level of complexity because all the people that are actually...Things were kind of working and nodding, that just cued up as just a whole level of disaster. It was kind of funny, in their postmortem they said, "We'll never do that again."

Gene: And what a great example of not only a gamble, but whatever ambiguity or imperfect understanding they had was immediately resolved when everything went to pot after that.

John: Exactly. Number twelve, human practitioners are the adaptable element of complex systems. Thirteen, human expertise in complex systems is constantly changing. Fourteen, change introduces new forms of failure. Fifteen, views of cause limit the effectiveness of defense against future events.

Gene: I love this one because of what you talked about before. Write down whatever's viewed as a cause is really based on a mental model. An imperfect mental model that we are imposing onto a complex system. Which is by definition risky.

John: And, therefore, it is wrong.

Gene: Wrong

John: Number sixteen, safety is a characteristic of the system and not their components. Safety is a human process. How we think about safety. How we apply safety. It's a human process.

Gene: I'm even seeing this phrase: "Safety is an emergent property of systems."

John: Exactly. Seventeen, people continuously create safety.

Gene: So, John Allspaw famously said in 2017, he asked, "How long can your systems continue to operate if you didn't change anything? Minutes, days, hours, weeks, months, or years?" What was so amazing about that question is that really, in my mind, it showed that, yeah, there are certain category systems that can run for months or maybe even years, but for complex mission-critical systems in general, it takes continual heroics to keep that thing running.

John: Yes. Finally, eighteen, failure-free operations require experience with failure. You know, I think there's always been this embracing of failure, we have to get good at failure. We have to create that as our model. Instead of being afraid of failure, we have to constantly embrace it to get really good at failure.

Gene: Yeah, Dr. Steven Spear once talked about the need to reinforce the dangers and hazards of our work. Not to the point of paralysis but to make sure that we always have a sense of caution and wariness.

John, thanks for walking us through those eighteen points. I suspect that everyone listening to this will resonate with all eighteen points. There's no doubt that the work that we do every day is some of the most complex and dangerous work on the planet.

John: Yeah, Gene, hopefully what people will see is that they are actually living mostly in a Safety-I world and that they really need to start transitioning to a Safety-II way of thinking.

Gene: So, let's talk about some of our favorite examples of technical and architectural countermeasures. I think one of them, without a doubt our favorite, is called the circuit breaker pattern. From Mike Nygard's famous book *Release It!*.[8] So, there's this problem that we've all faced where small errors, like calling an external API, fail or time out, and they lock up the entire program. Mike Nygard defined the circuit breaker pattern to ensure that these type of small errors result not in large problems but small problems.

John: Yeah, you know Adrian Cockcroft, who was one of the primary architects of the Netflix infrastructure, their cloud transformation, all those things. When you ask him, "Adrian, how did you build such a great architecture?" he very simply says, "Oh, it was easy, John, you know. I took one book by Nygard called *Release It!*. I gave it to the developers. I took another book from Dr. Sidney Dekker, *Drift into Failure*,[9] and I gave it to operations." And almost, like, "Okay, I'm done." But to look at...if you look at some of the characteristics of Mike Nygard's traditional circuit breaker patterns, basically, see the idea of wrapping and protecting a function in what we call circuit breaker object. The metaphor is like your home: You trip an outlet or a circuit, you don't lose electricity in the whole house. It gets isolated. It's based on that idea in software.

You basically...it's got built-in monitoring, the model would have a built-in monitoring system inherently in it. It would protect against latency and control of latency and failures from dependencies. Unlike you, it addresses cascading failures. The idea is use these patterns, and then you mitigate

against the cascading failure. And it just drives...when you use these patterns it drives failing fast and applying rapid recovery. You get fallback, graceful degrade. It's all built in to this idea. And really, in the end, it's about isolating access points between services.

Gene: Nygard talks so brilliantly in this book about how you have to proactively design your failure modes; otherwise, the failure modes will be in ways you've never imagined and you'll discover them in production. And he writes so eloquently about how it's really like bulkheads in a submarine. You really want to make sure the failures are contained.

By the way, when I first read Nygard's book, it was really painful to read because he'd find so well what goes wrong, and his patterns are so brilliantly simple, that you can't help but kick yourself while you're reading the book.

John: Absolutely...In this whole work, we've been talking about people who are giants and introduce things. And again, if you look at Mike Nygard's work in this simple book *Release It!*, if you look at today, or even just leading up to today, you had Netflix who created a whole open-source project around historics. Now historics is actually in Pivotal Cloud Foundry, it's showing up in tons of places. The actual commercial version of EngineEx, the proxy, will include now circuit breaker patterns. And even bleeding edge right now...the work that Google is doing on some of their proxy in collaboration with Lyft, a project called Envoy. Google basically has blanket created this thing called Itzeo project. It's all based on circuit breaker. You can't turn around the corner without getting hit in the face without circuit breaker pattern.

So, and even if we go and look at, in our world, at the DevOps Enterprise Summit. We had Key Bank, we just call him John Rez, but he's one of the leading thinkers there. He shows up in 2015 with this curiosity about, what is this DevOps? Less than a year later, he submits a paper [on] how they're

running a production banking application with containers, container orchestration, but more importantly, absolutely embracing circuit breaker patterns.

And I was able to actually recently visit their shop. They're running Netflix historics. And you see it, and I actually got to see a denial of service while I was there that tripped the circuit breaker, you sort a denial of service on the screen, and they'd basically mitigated and isolated it, even visually and through the architectural practices.

Gene: That's such a great example, something that was once obscure but is now absolutely mainstream. Here's another one of our favorite examples: game days. John, where do game days come from?

John: Yeah, so game days originally showed in an *acm*[*queue*] paper in 2012. It was titled "Resilience Engineering, Learning to Embrace Failure."[10] It was actually a discussion with Jesse Robbins, who at the time was at Amazon. I had the fortune to work under him in Chef when I was working a start-up. Kripa Krishnan was over at Google. John Allspaw, you know, here again John Allspaw. And another good friend of DevOps, Tom Limoncelli, who actually was Google SOE. Now he's at stock exchange.

But I think Jesse tells an amazing story about how he applied game day at Amazon, and he's publicly stated this. I just love this story, and I love retelling it.

Basically, the idea of the game day at Amazon was that they were literally going to take down a region or one of the data centers. They gave the forewarning that it's probably going to happen on a Wednesday or a Thursday. I don't know if they got it down to the day, but literally everybody knew that this was going to happen. And sometime during the day they literally cut off the core routers to this thing so nobody who interfaced with this whole data center could get to it. And as Jesse tells the story, there are

about six hundred people on the phone, and they're basically yelling and screaming for Jesse to turn it back on. You have to understand that Jesse, he was an EMT, he actually worked for the Seattle Fire Department before he became a rack and stacker. And so, he was at Amazon early in, and now he owns all commercial properties. In fact, his title there, his self-named title, was Master of Disaster.

And so, people are like "Okay, Jesse, we get it. We probably have to do XYZ to fix this. Please turn it on." And Jesse's like, "Nope, it's blown up." And they'd be like, "Jesse, knock it off. Do I have to call Bezos?" And in the classic fireman way, he'd be like, "You can call Jeff Bezos, but I'm going to tell him its blown up." And I think the core messaging there was Jesse wasn't going to let them table their fixes. They had to live through it. They couldn't be just like, "Okay, we'll save this for later. We'll fix it." They were going to force them to go through the real...he was not going to let go of the fact that it's not here, folks. You know, you have to figure out how to deal with this right now. This isn't fantasy world, where if it really was blown up we'll just turn it back on and we'll let you fix it tomorrow.

Gene: So, I love the picture that it paints. Here's Jesse Robbins, who stages a real live drill. Data center blows up, and I can—I think we've all been in organizations where you're halfway through this and you realize you can't get the data center back up. But now is not a really good time to have a full-scale drill, can you just please turn it back on. And I think we've all been in places where we said, "All right, we'll save ourselves a bye. Let's try it again next week." But there's never a good week for this type of work. And I think the reward, by having these sort of disaster-scale fire drills, is that they ended up with this world-class resilient service.

John: Yeah, and there's a theme there of...If you go back and watch any of Jesse's presentations, the first thing it'll show you is him in his full fire

department gear. And from that world, when you were simulating a disaster, the person who was playing dead was dead. They didn't get up and say, "Hey, I'm getting coffee now." So, it was really that fireman's mentality of how you deal with these kind of disasters.

Gene: One of my favorite stories that was also in the *acm*[*queue*] paper came from Kripa Krishnan. This was all about the dirt exercises at Google, very similar to what Jesse Robbins describes. That was a disaster incident response team. And it was all about these massive scale exercises to simulate failures to make sure that they could actually come back up. One of the things I found so memorable about her stories was just what would go wrong that you would just never guess. Until you go through the drill, you would never surface these kind of latent failure modes.

One of them was that they found that when they put engineers on a call, a lot of the young engineers had never actually dialed into a telephone bridge before. They had instances where people would put the entire conference call on hold music and there was no way to disable the hold music. Some of them were more significant. They found that data center managers didn't actually have the payment mechanisms to pay for more diesel fuel for the backup generators. So, through these simulations of not just data center failures but I think famously they had scenarios where aliens were descending upon every metropolitan area, they actually did gain an incredible amount of confidence and said that their services could ride out some of the biggest disasters.

So, let's talk about one more of our favorite patterns. We got to talk about the famous Netflix Chaos Monkey. I mean this is—we had talked about this a little earlier about the famous big Amazon EC2 failure that happened on April 21, 2011, and how the world then learned about Chaos Monkey.

John: Save the best for the last. Probably the most dominant discussion about these kind of patterns for resilience has been Chaos Monkey. It's almost a cliché in DevOps, which starts at Netflix. It's an architect design of, "We are going to inject failure and chaos into our infrastructure on purpose." Here again, one of the questions I had asked Adrian Cockcroft early on about how do you get into this chaos injection? And we'll talk about some of the patterns about how it started and how it's evolved. But, he said it was very simple. He said, "Here's the documentation about how you should be doing things." They didn't tell the developers they had to do things. They said here's the pattern, and you know circuit breaker patterns, bulkhead, all those things, and they gave them pagers. And the ones who found the patterns are the ones that, when they got an event in the middle of the night, it was an isolated failure. Sometimes they can go back to bed; other cases, they can fix really quick. This is Adrian's discussion… The ones that didn't figure out pretty quick that instead of spending four hours every night trying to track down cascading failures I might want to take a look at that paper.

Just to go through, it started out with something just called Chaos Monkey. Chaos Monkey, simply...Netflix was running on an Amazon infrastructure, so all their virtual instances were, basically, their compute instances, if you will, with virtual. They had this system that would just randomly kill a server. You had to basically build around the fact that not only will this happen in the real world at some point, it's probably going to hit you sometime randomly.

They built around that, and that just creates resilience. That was kind of host space. Then they grew that into a gorilla. So, they adopted some of Jesse's idea of taking out a whole data center, moving on to—I think they called [it] something [like] Kong—let's take down a whole region on Amazon. They point that out to things what they call Latency Monkey. Forced latency—find out what happens when you exceed the hypothesis that

you're going to see a lot of people's default timeout values. Let's see what happens then. Conformity Monkey—the list goes on, but these are so important because they've just matured...By the way, this is all open source...

Conformity Monkey, things that you better have tagged, you better have a tag from where it was in source control. The lineage of how it got there...You will get taken down if you just randomly put code in and haven't followed the process. Then they've explored Security Monkey.

One last thing that doesn't get mentioned as much as the Chaos Monkey —it's a subtle difference but it's so interesting. It's something they call FIT, fault injection technology. Along the same line but this is more of a...we're going to put inside a deep transaction, we're going to leave breadcrumbs...this is a high-level explanation of FIT. It will leave breadcrumbs all the way down, and then we're going to blow up that transaction. So, you get to actually see not only the kind of dump of what the transaction looks like when you purposely blow it up, but you get to see how it got there with its path. Such an incredible and interesting tool for complex systems.

Gene: I love how you just went through the lineage of Chaos Monkey, turn into Chaos Gorilla, turn into Chaos Kong, and the other members of what they call the Chaos Simian Army in Netflix. One of the ones that I found very surprising was one of the properties of Conformity Monkey. They have a service catalog where they register all the services, and one of the most astonishing things they do there is look for any services that don't have an escalation field with a developer's email address. If that field isn't filled in, they kill the service. So, the logic is, you find out in a planned way, in the middle of the day, as opposed to finding out in the middle of the night where you have to wake up everyone because we don't know who owns the

service. It's just another great example of how much thought they put into the Simian Army.

We've talked about architectural and technical patterns, which are always so much fun to talk about, but there's an equally vibrant and an equally critical branch of work around what is called Just Culture. This is a little bit embarrassing, but for years I actually misread the title of Dr. Dekker's book. I thought it was, "It's Just Culture," as in it's only about culture. I totally actually even missed the scales of justice that were actually on the cover of the book. So Just Culture really means, it signifies justice or judgment as in what is morally correct.[11] John, tell us about that.

John: Yeah, we've talked a little about that. Dr. Dekker has a little bit of a sense of humor. I think there's this...a head fake and a head fake, like this "Just Culture." It's not just culture; there is just culture in that if you do something wrong I'm going to fire you, or there's just culture that there's a balance.

I think we should take a second and look at Dr. Dekker's resume here for a minute. He basically, as we said earlier, he studied under Dr. Woods. He's currently a professor at Griffith University in Brisbane, Australia. He founded something called the Safety Innovation Lab there. He's an honorary professor of psychology at the University of Queensland. He's written a numerous amount of books. We've talked about a few of them; there are many more. *Field Guide to Human Understanding*,[12] *Just Culture, Drift into Failure*...all these books that we've referenced. He's got a new book out, it's called *The Safety Anarchist*.[13] It's a good work because it's combined his years of expertise. Dr. Dekker describes and talks about a just culture; what he's really trying to say is there are basically two ways to think about this. There is the kind of retributive just culture, which asks which rule is broken? Who

did it? How bad was the breach? What should be the consequences? Who gets to decide this?

Or there is a restorative just culture. Who's hurt? What do they need? Whose obligation is? How do you involve the community in the conversation. It's a clearer way of looking at Safety-I, Safety-II, but telling us a little bit that there is an equilibrium between the balance, the justice scale.

Another area that I've been fascinated by in Dr. Dekker's work is something he calls the Bad Apple Theory. Again, he's a creative cat, this guy. He goes back to Adam and Eve. Eve grabbed that apple and everything went wrong. You talked about his Brisbane presentation. I mean, he actually does a history of safety where he talks about the Bad Apple Theory and where we've tended to blame people. What he's really trying to get across is to understand that the way we look at things...we tend to fall to this Bad Apple Theory, where complex systems are basically safe and they need to be protected from unreliable people—the bad apples.

Human errors cause accidents. Humans are the dominant contributor to more than two-thirds of mishaps. Errors are caused because of loss of situational awareness, complacency, and negligence. The situational awareness is interesting. In IT, this is another memory muscle, we think of situational awareness as we evolved in managing systems, the big knock with all the screens. I think John's got situational awareness of everything that's going on. What Dekker and a lot of these guys would object to that notion is kind of the opposite of it. "Gene, you've got situational awareness at the data center, and what'd you do, you drank your coffee and you lost situation awareness." So now, you're the one who broke the system.

Dekker will get furious when he sees an article about "the pilot lost situational awareness;" therefore, the plane crashed. It's just not that simple. Errors are introduced in the system only through inherent unreliability of people. I think you can wrap this back, we've talked about Safety-I and

Safety-II; we've talked about leaving that notorious Frederick Winslow Taylor. You can see this inherent Bad Apple Theory throughout the thread of the work of Taylor. Another thing that one my favorite all-time Dekker quotes is the...He says that Murphy's law is wrong. He says… this is actually referenced in *A Field Guide to Human Error*...he says, "What goes wrong, usually goes right." What can go wrong, usually goes right. But then we draw the wrong conclusions.

You talked about this earlier. This idea that there's a lot of things that that engineers do a great job of. Those are the things that are actually going right that we don't get to see. I think, finally to summarize all of Dekker's work in a very simple way, I think that what probably keeps Dr. Dekker or Dr. Woods, Dr. Cook, and John Allspaw up at night is the idea that when you're doing a retrospective on something, like a horrible accident at the hospital, like a baby dying. How do you get to a safe place where a nurse or a dietician who are involved in the thing could easily, in a Bad Apple Theory, be blamed for that? How do you create an environment where they'll help you explain the real system's problem to what happened? I think that's why you'll hear a lot of these authors get upset about a root cause or situational awareness. They realize those are things that lead right to "Oh, it was the nurse." By the way, the nurse is afraid that she's going to get fired because of what could've happened.

Gene: Dekker would say that we need that nurse to be a part of that solution. In fact, maybe only the nurse can actually create the solution. And I think what I love about this, and you've helped me have some aha moments around this, I think anyone who's been familiar with *The State of DevOps* report that we've done with Puppet and DORA over the years, one of the models that we introduced was the Westrum organizational topology model.

So, Dr. Westrum famously divided organizations into three types: pathological organizations, bureaucratic organizations, and generative organization. And, to vastly oversimplify, pathological organizations [are] always looking for someone to blame. Bureaucratic organizations, they wanna be just. In the retributive just. Whereas generative organizations are really the ones where there's a common objective, and they know that results of accidents and failures create a sense of genuine inquiry.

So, this takes us to something that all of us should be familiar with. It's the famous blameless postmortem, or sometimes called blameless post-incident review. John, many of us might not actually know where this practice came from. Where did it come from?

John: Here again we talked about John Allspaw's contributions, and many of them started out early as blog articles. There was an article that he wrote about just culture, which was basically taking Dekker's work and basically explaining how they applied it while he was at Etsy to their postmortems or after-incident discussions, you know?

We call them postmortems or retrospectives. And what they did is… And actually, this is well documented in Etsy's "Code is Craft" blog article titled "Blameless Postmortems."[14] They looked at what are the actions they took at the time? What effects did they observe? What were the expectations they had? What were the assumptions they had made. The abductive reasoning...all these pieces...And then the timelines.

They forced themselves to look at a systems approach to all the things. And there's a whole sub-narrative of, "No, blameless is...You can't say, 'Well, Bob did it.'" But applying this kind of layering of forcing you to look at all these things and correlate them together...And they further went on to describe [how], "Here at Etsy, we run a just culture." So, it's just culture at Etsy, and you know what they say is, we encourage learning by having these

blameless postmortems on outages and incidents. We understand how accidents happen. And in order to better equip from it happening in the future.

They're giving the ammunition of how to do this. To think this way. The "gather details from multiple perspectives on failure. We don't punish people for making mistakes. We seek out second stories. We gather details from multiple perspectives." And we've talked about this mental model: everybody has a different mental model, so you have to literally just swath all of that together to get some semblance of truth. Or get to a place where you have a better understanding.

You enable and encourage people who do make mistakes to be the experts on educating the rest of the organization on how to not make 'em. We want to have a junior person come in, and we're not going to let...He even talked about in his recent DevOps Enterprise Summit, where this junior person was "All right, before we get started, I broke it. End of story." And the senior people, "No, no. That's not the answer." And [the junior person's] "What do you mean? I'm taking the blame." No, no.

And what they found is the junior person actually did really understand, and they really didn't break it. We accept that there is always a discrepancy space where humans can decide to make actions or not, and that the judgment of those decisions lie in hindsight. The judgment lies in hindsight.

This is probably the one...This one I really love. "We accept that hindsight bias will continue to cloud our assessment of past events and work hard to eliminate it." They're not fooling themselves. People are "We don't have any cognitive biases here." No, no. It's here, we know it is, and we're gonna accept it and embrace the fact that we know it's always looming over us. The same thing with this… Another really interesting cognitive bias is this Fundamental Attribution Error. It...So, they say, "We accept that Fundamental Attribution Error is also difficult to escape, so we focus on the

environment and the circumstances people working on when investigating accidents."

In short, the Fundamental Attribution Error is very simple. "I think Bob did it. Bob is a messy coder, so for sure this incident has to be because Bob is always...his code is always ugly." It's to break yourself out of that. You know, some other points. "We strive to make sure that the blunt end of the organization understands how work is actually getting done."

We've talked about the sharp edge earlier. "The sharp is relied upon to inform the organization where the line between appropriate or inappropriate behavior..." This is something that the blunt end can come up with on its own.

And a couple more. "One option is to ensure that single-cause is incomplete and scream at engineers to make them pay attention or be more careful. Another option is to take a hard look at how accidents actually happen, treat the engineers involved with respect, and learn from the event."

Gene: One of the people, I think, who's been so wonderful in articulating the culture at Etsy is Bethany Macri. She created a tool called Morgue, morbidly, to sort of manage the post-incident review process, and she said something I thought was so memorable. She said, "Prevention requires honesty, and honesty require the absence of fear."

But there was this other amazing Bethany Macri story that happened at DevOps Days 2013 in New York, which just shows how different the culture is at Etsy.

John: You know, when Etsy writes that "a just culture at Etsy is this." They throw it in your face. But what really resonates is...I'm gonna throw out a word: kata. How do you create that persistent memory muscle so that it almost goes out...Like katas, like kabuki theater, or karate, where you're

gonna make a motion because you've done it so many times you don't even think about it.

And the thing I love about that...It was in the New York City DevOps Days 2013. She gave a presentation on that postmortem tool that she wrote, and in the Q&A there was all sorts of questions about, "How does the tool work?" And so one gentleman asked Bethany, he said: "How do you ensure that somebody actually follows up on the assigned remediation action?"

And, I've talked about this in presentations. You actually have to go watch this video, it's at the end...At first, she is bewildered. She doesn't understand the question. And at one point she says, "I don't really understand your question." And then she's thinking about it a little more, she puts her finger to her head, it's just so theatrical, but it's such...It shows how they had built this just culture that she goes, "Oh. Yeah, that doesn't happen at Etsy."[15]

It was like she couldn't understand. I'm gonna say one thing. So, just to steal a little bit of Lean back into here...There was this story about how when the American people came to look at the manufacturing, they noticed that when the door was put in, they didn't have to use one of those rubber mallet hammers to bury it in. And at some point, they started asking the question, "How come you don't use a hammer?" And the workers were "What are you talking about? A hammer? Why would I use a..." And then they were "Oh. No, no, we figured that out way earlier in the process."

You know? It's just a culture that, when you know you've got it so right, you don't even understand these obscure questions.

Gene: Yeah, in fact there was a wonderful...I think she finally closed up the question by saying, "Yeah, we don't close the incident until the remediation action is complete."

John: Yeah, yeah. That's brilliant.

Gene: That's awesome.

One of my favorite stories about blameless postmortems is from Randy Shoup. For many years, he was the engineering director at Google for the app-engine product, and what was marvelous about his description of it was just how powerful of a mechanism it was for organization learning, and he said a couple things. One was that even having public blameless postmortems unlocks engineering competitiveness. In other words, not only do you make it safe to talk about problems, but engineers love disaster stories, and whenever you...they shared out a big disaster story...there was always one engineer who would say, "Oh, you think that was bad? Let me tell you about something that happened to us four weeks ago." And then they would actually write up that postmortem. It is a self-amplifying mechanism.

The other thing that he talked about that just blew me away was this, is that the invariable outcome of doing these blameless postmortems is that you get safer systems. They genuinely prevent problems, and then the next problem, because there aren't enough postmortems because there are not enough customer-impacting incidents. One of the things that he did was conduct postmortems not just on customer-impacting incidents but team-impacting incidents. In other words, we put seven safeguards in place to make sure that bad things don't happen, and six of them failed. Let's have a postmortem about that to make sure that we never get into that situation again.

So just like the Andon cord pulls, we keep tightening up the tolerances to not only increase safety but also to ensure that the organization is always learning.

John: And think about the kind of transparency in postmortems. Another good friend of DevOps, if there's a group of us, Mark Imbriaco, he basically —I remember being at Chef at the time, and I worked for Jesse Robbins,

who had prior run, probably, the top three infrastructure in the world, and when Mark was at Heroku there was a major outage. Mark wrote this very long public postmortem where he explained everything and the dirty laundry. And I remember Jesse going, "That's how you run operations."

And what was interesting, I've convinced Mark over the years that that was the first time in web-scale or DevOps that I had ever seen somebody create this public truth postmortem to all their customers. Now, since then, we've had Amazon. Now everybody does it. But I think he introduced this opening up that when we talk about internal value in running postmortems, imagine you expose that to your customers and you gather aggregate information at the sharp edge, if you will.

Gene: John, we started off by suggesting that DevOps draws most heavily from two fields. The first was Lean, and the second was safety culture. And hopefully, everyone sees how much safety culture has influenced DevOps, both in terms of the technical and architectural patterns that we use as well as creating a just culture that enables not just blameless postmortems, but ultimately, organizational learning.

John: Yeah, Gene. I mean, when we look at the kind of arc of John Allspaw and his influence, we start off with him introducing how complex DevOps systems fail. He gives us this notion of how it can apply to web operations. He takes us on this journey with just culture and how Etsy applies these models. And then, we talked a little bit about his thesis, completed in 2015, where he actually took all the theory and applied it to a really large web-scale operation, and took the theory and showed us how it actually works for a publicly traded company.

And then, I think, recently, the 2017 DevOps Enterprise Summit, his presentation—

Gene: Yeah, he presented about his learnings, and what struck me was how he began the presentation. He said, "The last time I gave a presentation that I thought was so important was in 2009." So, what did he present?

John: So, he gave this notion, and there's actually really one really cool slide where he describes, in part of the work, he talks about an above-the-line and below-line framework. And he says that there's this notion of a system, and it includes its makers, its modifiers, its watchers, its compensators. And what's above the line is the people, the multiple people that deal with the system. Each one of these people have their own mental model and abstraction of understanding what the system looks like. It's their observation. It's their anticipation. It's their planning. It's how they troubleshoot, diagnose. It's their general understanding of how they operate. The key point is, it's multiple people with imperfect or different notions of what the system is above the line.

And then below the line is what we traditionally think of [as] what are you running? Well, I'm running this many servers, and I'm running this operating system, and I've got these database servers, and I've got this network architecture, and I've got cues, and I have monitors, and logs, and all that.

Gene: And it also includes the code, and the deployment pipelines, and all that stuff.

John: And so, he introduces this notion of a very simple above the line, below the line. And then, kind of drawing on all these notions and ideas about complex systems and all this does, he makes this observation that by the way, the below the line doesn't exist.

Gene: Right.

John: And then your jaw drops. And it, again, it reminds me, I was in the audience in the 2009 *10+ Deploys a Day at Flickr*, and I remember distinctly hearing that and going, *What did that young man just say? That cannot be accurate*. And I had that same feeling when he made that [statement]. Just, in your face, I'm sorry folks, what you think exists below the line doesn't exist.

Gene: I think both you and I struggled with this. We argued both with Allspaw and Dr. Richard Cook because that's just a heck of a claim to make. Everything below the line doesn't exist? But upon some reflection, it's not as crazy as it sounds. In fact, if you go back to Dr. Cook's 18 Points, I'm gonna read off a couple of them.

Number six is catastrophe is always just around the corner. Ten, all practitioner actions are gambles. Eleven, all actions at the sharp end resolve all ambiguities. And so, my takeaway from this is that if all those things are true—and I think they are—whatever grasp we have of whatever's below the line isn't actually that reliable, that our systems are so complex that our understanding is so imperfect because really, in reality, no *one* human being can understand the full scope of the system.

So, for all intents and purposes, whatever is below the line may as well not exist. And I think it gets to point number seventeen that Dr. Cook asserts: people continuously create safety. That is really what is above the line. So, I think Allspaw makes a very controversial claim that below the line doesn't exist. I think what we can all agree on, though, is that above the line is all that matters. I think that's what was the power of that provocative statement that Allspaw makes.

John: Yeah, and I think we've introduced the imperfect notion, or evidence of the imperfect notion. Cook's work, but even, why did we have Chaos Monkey?

Gene: Right.

John: If it was perfect, why did we have to create purposely failed embracing resilience? Why did we have circuit breaker patterns? We put circuit breaker patterns in because we accept the fact that these complex cacading problems will happen, which is, by definition, owning up to the fact that we really don't understand how a system works.

And I think that the story you told about Kripa Krishnan, there's an example where they apply a game day, or the dirt, as you described it. And they found out all these other imperfections that they had no idea was the things that you described.

Gene: Yeah, in fact, in that story they couldn't even comprehend exactly how telephone conference war bridge calls would work. So, I think that's very humbling, owning up to our imperfect understanding of how complex systems work.

John: So, imagine a telephone system as a box in below the line.

Gene: Right.

John: The thing that I think we both agree on is that today we take for granted what John was saying in 2009. We've got data from DORA and the study [*State of DevOps Report*] that tells us that high-performing organizations behave in a certain way. It's standard practice of high-performing organizations to basically embrace deploying fast, creating much more than ten deploys a day. Thousands. But back then, the notion was...I used to joke—and I think I have joked—people were throwing up in the back of the room. "How could he say that? How do you do that?"

And I think what we're trying to say here is that we fundamentally believe what John is telling us, again, and why he thinks it's so important. The

evidence is that we're gonna look back, and even though to most of us right now [we] would be "What is he talking about, it doesn't exist?" I think we're gonna find that this presentation will be very much like the 2009 presentation of *10+ Deploys a Day at Flickr*.

Module 6

Learning Organizations

Gene Kim: So far, we've covered Dr. Goldratt and Dr. Deming in Modules 2 and 3. In Modules 4 and 5 we covered Lean and safety culture, the two fields that DevOps draws from most. At the risk of oversimplifying, you and I have both concluded that there's absolutely something in common with all of these, and that is learning. There's a saying that goes, "The purpose of a learning organization is to out-learn the competition." John, what are learning organizations, and where did that concept come from?

John Willis: We've talked about Andrew Shafer throughout this work, and I think we both love one of his quotes, actually in our *DevOps Handbook*, where he says, "You're either a learning organization, or you're losing to someone who is." I think if we had to go back and look at the father, what most people consider the father of learning organizations, you would go to Peter Senge. Peter Senge, here again, another MIT Sloan guy, but his probably most famous prominent work is something called *The Fifth Discipline*, and that was actually originally written in 1990. Just to be

completely honest, not an easy book to complete. I think one of the jokes is that everyone talks about it but nobody's actually completed it.

Gene: And I, too, find it very challenging, and there's a book that's allegedly easier to read called *The Field Guide to Learning Organizations*, and I found that equally challenging, so you're not alone.

John: So, maybe we're challenged; I don't know. One of the things I love, when I bent back and started doing some research in earnest about learning organizations, and a lot of that driven by Andrew Shafer and some of his presentations that we'll mention later, but one of the first things I found—a little coincidence—is actually in the first edition of *The Fifth Discipline*. Peter Senge, by the way, Deming was still alive, he sent him a copy to say, "Hey, Dr. Deming, I would love for you to review this," and Dr. Deming wrote back a letter to him, and actually Peter Senge had admitted that he had actually changed some of the content in the second edition. But if you let me, I'd love to read Deming's letter that he sent back to Senge, which is such a classic Deming.

> You know our prevailing systems of management have destroyed our people. People are born with intrinsic motivation, self-respect, dignity, curiosity to learn, and joy in learning. The forces of destruction begin with toddlers. A prize for the best Halloween costume, grades in school, gold stars, and up on through the university. On the job, people teams and divisions are ranked—rewarded for the top and punished for the bottom. Management by objectives, quotas, incentive pay, business plans, put together separately, division by division, cause further loss, unknown and unknowable.[1]

This was almost like he was lecturing him, and the whole idea about people and how we let our systems degrade our people. But the other thing...One of Senge's really good quotes is this: "A learning organization is a place where people are continuously discovering how they create the reality."

Gene: Wow.

John: So, I wanted to try to summarize for us today *The Fifth Discipline*, it's five disciplines. One is personal mastery. We have to continually clarify our vision. It's that self-improvement, and we're constantly improving, which is a core to learning. Two, mental models. Right, we've talked about mental models throughout, in fact, in the safety culture and, basically, these are our assumptions, our generalizations, our pitches, our images, how we're influenced, and how things...how we think a system's...Another big part of Senge's work is systems thinking. Another one, building shared vision. It's basically trying to create a practice of people sharing and fostering the roles and what we do, and what we see, again, the mental models. Team learning starts with dialogue. The capacity of members of the team to suspend assumptions. We see this in postmortems. And probably the most important —and really a lot of people would think of this as a virtual cycle—would be system thinking be really the first one in the cycle, but it's an integration of the other four points.

Gene: Yeah, I love the enumeration of the five disciplines, and even in the building of shared vision there's a phrase: "We want to foster genuine commitment in enrollment rather than compliance." So, even there, we see echoes of Safety-I and Safety-II. Safety-II is not about compliance to process; instead, we need to focus on the goals of the system as a whole.

John: Yeah, no, absolutely. And then you know, as I was further researching, I found this really interesting article in *Harvard Business Review* in 2008 by

David Garvin and titled "Is Yours a Learning Organization?"[2] and what he was building on was work from Senge on *The Fifth Discipline*. And he said that learning has these building blocks, and he listed some of these ones, but the one's we've talked about, psychological safety, this shows up over and over. We've talked at one point about the Google work that was done. Google did tons of research, and they found that psychological safety was the core to building great teams. We see that over...Appreciation of differences, appreciation for diverse groups to help resources, our cognitive biases.

Gene Kim: Yeah, I love that, and I think the modern...with the thinking fast, thinking slow, I think the modern treatment would be the reason why we need diverse teams is to help overcome all the cognitive biases that we all bring.

John: Yep, absolutely. And open us to new ideas. Basically, they all kind of tie together—psychological safety, appreciation of differences, openness to ideas—and that implies anybody can have this new idea. You can be the junior person on the team, and you have this psychological safety to be able to say, "Hey, I've got an idea," and a high-performing team would be "Let's hear this."

But one of the things I thought was interesting in our recent conversations with Dr. Spear, the concept of psychological safety came up, and I think this all ties together. Where he said we need to be careful. We're not just talking about the edge of the line work; we also need to understand that the leaders have to have psychological safety.

Gene: Yeah, I think his words were, "There's a stunning level of humility required in the leader to be able to say, 'I don't necessarily know what to do. There are some things I don't know enough to make the right decision,'" and

then he said, “When leaders act like pompous jackasses, it negates whatever psychological safety the employees felt.”

John: Yeah, and imagine the comfort of a leader saying, “You know what? I’m lost. I really don’t know how to answer this question. Time for reflection.” That’s a big part of what we constantly talk about.

Gene: Yeah, I think in the development world, that’s what we call the “slack time,” and I think in the Lean community would say, “These are the times we dedicate for improvement of daily work versus just daily work itself.”

John: And then systemic knowledge sharing. I love this. In fact, if we go back to that concept we talked about in the last section about above the line, below the line, when we talk about different people having to get together to share mental models, which by definition can be imperfect and different, what we need to concentrate on is getting really good at creating a systemic way for us to share our knowledge.

Gene: Love it. Dedicated for improvement of the understanding of the system.

John: Absolutely. Education, experimentation. Education is table stakes in a learning organization. But education that drives experimentation or hypothesis-driven education, we’ve gone back to Deming’s PVCA; that’s the key. It’s not just “Let me send people to classes,” it’s teaching people how to think.

Gene: Eric Ries, *The Lean Start-Up*—he knows all about organizational learning to win in the marketplace.

John: And just reinforcement of those learnings. And in fact, we've talked a little bit about kata, kata is one mechanism, like Mike Rother's *Toyota Kata*, improvement kata, how do you...Okay, so, we're doing good. How do we create this reinforcement?

Gene: And even without going deep into any of these areas, what I love so far is that all these terms sound so familiar. We've touched on almost every one of these areas in the previous modules, and I love how… It gives me great comfort to know that organizational learning is all about themes that we've heard before.

John: Yeah, and then I would say that that is the thing that I think we...At the end of the day, they all have these traits where it is organizational learning. In my experience studying Peter Senge, one of his core ideas, his thinking was about the importance of fast feedback. He has his famous example of the temperature in a thermostat. If the feedback loops are slow, we overcompensate. I think for me, I think about all the travel. I'm always in a different hotel, so I have this...my mental model if you will...of what I want the temperature to be, and then I play with the dial. And I know over the years not to go right into the shower because if I turn it, it's either going to be scalding hot or way too cold. I think it's a great example of the feedback loop that I get is extremely latent from the time...I know it's a wait.

But you have a really, really interesting story that applies very well to this idea.

Gene: Yeah, I think one of the best examples of how slow feedback loops kill is a famous exercise called the MIT beer game. This was created...again, MIT Sloan...in the 1960s, and typically it's run in three or four players. It takes one hour to complete. And I remember actually playing this, my goodness, like fifteen years ago. The game goes something like this. If there's four

players, there's a retailer, a manufacturer, and in between them you have a distributor and a wholesaler. And typically, it's played ten or fifteen turns, and the rules are very simple. In each turn, the retailer sells some amount of products, and then every person has to order from their upstream partner. There's a couple of twists here, though. There's a two-week latency between when the order's placed and when it's actually fulfilled. Each player is penalized one unit for any excess inventory they hold as well as for any unfulfilled orders.

The results are pretty amazing. Even given the simplicity of this game, by many accounts across thousands of trials spanning decades almost everyone is out of business by turn ten. In fact, there's this amazing statistic that shows that CEOs of Fortune 50 companies perform no better than high school students. Regardless of the seniority of the people playing, when people interview them at the end, people express an incredible sense of frustration trying to figure out what is going wrong. There's an intense feeling of frustration because they're not getting the results they want, and they always feel like someone on their team did not understand the game or properly account for the fact that what matters is customer demand. They're trying to explain what is behind the erratic pattern as backlogs mount or massive inventories accumulate.

One of the interesting things that has been observed about this game, it takes only one person to over-order or under-order before a catastrophe strikes. Basically, the bullwhip effect, where the backlogs or the excess inventories skyrocket. They seem to go exponential.

There's a couple of observations that I think are relevant to what we've been talking about, is that this is not necessarily a complex system. There are four nodes or four players; there's three edges between them and only a handful of rules. And according to the Allspaw above the line, below the line, we can actually see what's below the line. And yet, it has so many

properties of what Dr. Cook talks about, catastrophe's always right around the corner, that these complex systems, even this simple system, are intrinsically hazardous.

John: Yeah, I was going to say that's a great example of...by definition it does not look like a complex system. So, we would think going into it that this is not a complex system, and by definition it really is a complex, adaptive system.

Gene: It's so interesting. We take this much smaller system. Imagine how hard any modern technology service is that has hundreds or thousands of servers, millions of configuration settings, hundreds of engineers, where if you count the number of nodes and edges it is orders of magnitude more complex.

This really gets to my last observation that, despite this...You know, I happened to go to the store yesterday, and I noticed that the retailer was still in business. The shelves were full of goods, and the suppliers somehow also didn't manage to go out of business. And I think it shows that...Dr. Cook's point is that people create safety in systems. Despite all these difficulties, people manage to...allows to compensate for all the hazards in the systems. And that learning organizations are a part of this. We learn how to compensate for this.

John: You know, Gene, it's brilliant. It's just...that whole thing, that thread that we've been talking about throughout almost all the information we're working on. When we talked about in the last section, about the five disciplines of mental models and systems thinking, and then, just to go back to that point about the building blocks, the one...my favorite one was hearing a systemic sharing of knowledge. That's how we're able to learn and

get better with these systems that, as Dr. Cook said, they're kind of hazardous systems.

Gene: Maybe one interesting point. As I was restudying the MIT beer game, it made me realize to what extent Walmart has totally changed the game. They've basically said to the entire supply chain, the only thing that matters is how many of your goods are on the store shelves. And the whole system is designed not for the local optimum of getting your goods onto the store shelves. In some ways, they massively simplify the problem and end up with a far better system in so many ways.

It's interesting to read the literature about the learnings from the MIT beer game. I pulled out a couple of them. One is managers need a cool detachment to be able to correctly diagnose and fix the problems that the beer game creates. Two, it shows the nonlinear nature of industrial and economic changes. And it shows the futility of blaming employees for problems beyond their control. And that operations must be managed as a system, not as set of isolated activities.

In another one of those interesting quirks of history, there's also someone whose work Dr. Peter Senge's work depended upon whose name may sound familiar to many of us, and that's Dr. Chris Argyris. He's professor emeritus at Harvard Business School. Tell us about him.

John: So my first introduction to Argyris is actually a presentation by Dan North, a good friend of DevOps, as I'll say over and over. If you don't know, Dan North is actually one of the originators of behavior and development.

So, we're both big fans of Dan North. He [Argyris] gave this presentation, and it was interesting because there's this idea called the ladder of inference. And what Argyris was saying is if me and you were having a conversation, I'm just synthesizing everything you're saying for me. Right? And what he said is, it's a ladder of the process of information that you get. So, you're

doing the same thing. So, he says, at first I'm going to observe this information, and I'm then going to selectively tease it out, and then, at some point, I'm going to give it meaning. And then I'm going to start assumptions about the meaning of it. And then I'm going to draw conclusions. And then, over time, there's actually a reinforced feedback loop in the middle here, where, over time, I see this over and over, they become beliefs. And then ultimately, I feel so strong that I'm going to pull out the poster and start protesting, or do whatever I have in actions.

And I love this because I think we're both fans of David Foster Wallace, and he wrote a book called *Infinite Jest* but there's a...they call it the most famous commencement speech. It's called "This Is Water."[3] I highly recommend it. You know, it's on YouTube. There's brilliant stuff in there, but one of the points he makes in there is about this same point, though he doesn't reference Argyris. He says we just tend to think of ourselves, and what we really need to do is to force ourselves to break out of those kinds of patterns or understand that we're in those patterns. He gives this example about this woman is driving an SUV, and she's cutting in and out of the highway, and your first reaction is, "Boy, gas-guzzling jerk." And then he says, what if you step back and said maybe she's on her way to the hospital. And so what Argyris says is when we can understand this ladder of inference...if we don't, we'll make bad judgments. Our assumptions can lead to bad conclusions.

But what we need to do is to question our assumptions and our conclusions. And that middle feedback loop...that area where we're basically putting meaning and assumptions and conclusions and beliefs. If we could force ourselves and just—coincidentally, that's what we try to do in blameless postmortems. What we need to do is break those conclusions, seek contra data, look for alternatives, force ourselves to look for alternatives. And make

those assumptions visible. Invite others to draw out conclusions. And ultimately, we're breaking that chain of bad decisions.

I've done presentations on this. I have this example I love. Imagine I'm speaking in front of an audience of people; it's in a company and there's a bunch of divisions, and I've got this new idea. And there's this one team, and their reinforced conclusion is against my team's idea. So, I'm giving the presentation, and I observe this person in the front row, they're constantly reading...at least for me, it looks like they're constantly reading their phone. And then I select, *Oh, it's that person*. Let me put some meaning to it. They're *that* team. They don't usually really care about what I'm saying. They're not listening to my presentation. And then, surely, he or she is not interested. They don't like my idea, which is my conclusion. My belief is that's that team that always blocks us. And then after the presentation, I send a nasty email to their manager. That's the bad example of falling into the trap of Ladder of Inference.

Alternatively, second scenario, same situation. I notice this person in the front row, observe they're fuddling around with their phone. The meaning is they don't like my presentation, but now at the assumption point, I'm going to go ahead and engage with that person. And I'm going to say, "Hey, did you think that was an important point or not?" And to my happiness, they say, "Oh, actually, I was rescheduling my next meeting because I wanted to see the rest of this presentation."

Then that drives me to a conclusion they do actually want to work with me. And I don't wind up sending that nasty email after work. That's a great example of [how] his Ladder of Inference is about making sure that we're just questioning what could be those natural feedback loops that he calls the Ladder of Inference.

Gene: You know, I actually saw the Ladder of Inference model being used when I was shadowing Scott Prugh, the chief architect VP of product development and product operation at CFG, who we've mentioned numerous times in this. And it was a day I was following him around during one of their release planning days, and we're sitting in a meeting, and there was this one question that came up around a certain feature. We're going to add one more step, which is to have some third party certify our code. I remember seeing Scott sort of immediately start leaning in and start going down the Ladder of Inference, and one of his conclusions was we're so much in the habit of other people certifying our code when, really, the people with the best expertise and people most able to certify code [are] ourselves. So, by revisiting those assumptions and breaking that bad habit and putting a spotlight on it, they were able to reduce a number of dependencies to actually get this thing to market. And it was just really amazing to see Scott Prugh very deliberately use the Argyris model to help coach people to make different decisions.

John, one of the things that I think is super interesting about the things that we've been discussing is the overlap with some of the learnings from the Lean community. John Shook wrote about his NUMMI experience in the Fremont plant, the joint venture between General Motors and Toyota. He wrote, "What my NUMMI experience taught me that was so powerful was that the way to change culture is not to change first how people think, but instead to start by changing how people behave, what they do."[4] Those of us that try to change our organization's culture need to define the things we want them to do, the way we want to behave and want others to behave, to provide training, and then to do what is necessary to reinforce those behaviors. The culture will change as a result. This is what I meant by it's easier to act your way to a new way of thinking than to *think* your way to a new way of acting.

So, what resulted out of that is the Shook Change Model. The old way is to change thinking to change behavior. Instead, he proposes it is better to change behavior to change thinking. What I think is very interesting is that the organizational development community also came to the same conclusions. Edgar Schein, a contemporary of Dr. Chris Argyris, has a similar model where there's a one-to-one mapping between culture and basic assumptions, as you'd described, John. Values and attitudes. And what we do is what John Shook said Schein would just call artifacts. So there is a one-to-one mapping between the John Shook model and the Edgar Schein model in organizational development.

John: Gene, you know that reminds me, going all the way back to, I think around 2009/2010, there was a local DevOps meetup out in Silicon Valley, and this gentleman, Lloyd Taylor, he was actually ex-Google, but he gave this amazing presentation about basically culture. And I'll never forget one of his quotes here is that "You can't directly change culture, but you can change behavior." Because we had been talking about culture this and you got to do culture that, and he was getting right to the point that we're talking about right now. It really takes the behavior to drive the change of culture.

Gene: And in fact, if my memory serves right, he actually said by changing behavior, behavior becomes cultures.

John: Yes.

Gene: Which is just a wonderful way—

John: Oh yeah, absolutely.

Gene: —of describing what culture is.

John, it's interesting that so many of the concepts behind organizational learning come from military, and many of these names are actually very familiar to us.

John: Yeah, we're always looking for how do we find new knowledge and force around learning, and in the military we have great examples of feedback loops. Speaking of the feedback loops, you could go back to a gentleman named Norbert Wiener, who was the creator of something called cybernetics. He was working on anti-aircraft shells, and the idea that the cybernetics was constantly correcting the guidance of the missile. So that making sure that corrections were just being completely filled back into the system or feedback into the system.

Probably the most famous, the one that's been discussed the most in the DevOps community, is John Boyd, who's famous for something called the OODA loop. The OODA loop is observe, orient, decision, action. John Boyd, interestingly, was a fighter pilot trainer who's actually a fighter pilot. It's funny that he had no kills as a fighter pilot, but turned out to be one of the greatest trainers of fighter pilots. He wrote a [paper] back in 1961 called *Aerial Attack Study*. And here again, what he was training other fighter pilots was that in order to win, we must operate at a faster tempo or rhythm than our adversaries. But we need to get inside the adversaries' head, and this is where you really started defining this OODA loop concept. The idea of the observe, orient, decision, is these type of activities, when successful, will make us feel or look ambiguous or unpredictable. It generates confusion and disorder among our adversaries.

In some ways, people have compared this to the PDCA—plan, do, check, act. It's, again, another one of those reinforced learning loops where you're constantly working on the feedback.

Gene: When you said that, it just immediately reminded me of this meeting I was in many years ago. It was a large software company in the educational market, and we're talking about how to incorporate DevOps. In the middle of the meeting the leader walked out and then came back in twenty minutes later and said something I'll never forget. He said, "And now, more than ever, this is more important. This is the most important thing ever. We have small competitors who are beating us everywhere. We just lost our top two accounts to this competitor. How can we, with ten times the size, not be able to compete?" And so, this is a great example of how smaller competitors were able to get inside the decision loop of a much larger company.

John: Yeah, and you know, if you look at what Boyd would say, it applies to what Andrew had said about learning organizations. He'd say that striving to operate at a faster tempo to generate rapidly changing conditions that inhibit your opponent from adapting or reacting to those changes...Not that everything is a warfare metaphor, but like Andrew said, it's building a learning organization or you will be losing to another one.

Gene: And I just remember the feeling that was being exuded by this. Our enemies are everywhere. We're being out maneuvered by them. How is it that we are losing to them? It was something I'll never forget.

John, another thing I find so interesting is that it seems like learning organizations have almost entered mainstream culture. When you walk through airports, you also walk by a bunch of bookstores, and I can't help but notice that so many of the best-selling business books of this decade are all about creating learning organizations.

John: And I think one of the things we often talk about is how this DevOps community is really good at sharing different books, particularly books that you wouldn't normally think are IT related. And you've recommended

books to me and others, and the community's just really good at sharing books. And you're right. All the ones that I've had the opportunity to read tend to just emphasize all these principles that we've been discussing about learning organizations. One of them, David Marquet, Captain Dave Marquet, called *Turn This Ship Around*.[5] One of the things that he expresses is this idea of intent-based leadership, where you give the intention instead of deterministically telling people what to do, you give them the intent of what can happen. And he uses his learning...he was captain of a nuclear-powered attack submarine, and he was unexpectedly changed to another class of service. He just didn't know all the features and functions, all the things that he had mastered in the previous class, and he had an inspection coming up relatively soon. So he was forced to use a different model where he couldn't tell people you have to do this, this, this, and this. He had to change, and I think if I had to summarize or say, he had this aha moment about this model of intention works much more effectively.

Gene: I love that story because he was trained on this totally different type of submarine, and he couldn't give orders to do this, this, and this because he didn't know. He didn't have the expertise or the background, and what I love about that story is just the results of that.

It went from one of the worst-performing submarines in the fleet with the lowest performance levels, highest failure rate in inspections, lowest retention rates, and by some number of years later, it had the highest levels of retention and the highest levels on inspections. It was setting the standard for the fleet. Just a phenomenal example of just how these things can work even in a highly regimented organization like the US Submarine Corps.

John: Yeah, absolutely. You know, another book that really is, I think it's core to even the DevOps movement, is Eric Ries's *Lean Startup*. In the early days, a lot of the convergence of DevOps, some of the core pieces, came from that

book. That book today is actually used in colleges for business start-ups, whether it's IT or not. But basically, and we've talked a little bit about it, he had the build-measure-learn, the pivot. It was, again, another model of making sure you're breaking these patterns of old businesses, where you had to be constantly adapting and working off of feedback. It's no coincidence that, at the end of that book, he does attribute a whole section to Dr. Deming. Just saying.

Gene: That's just so great. I think what's interesting about the *Lean Startup* book is that Eric Ries has actually introduced a new language, which is now being used in business, which doesn't happened very often. I think that's really a remarkable testament to the power of his ideas.

John: Yeah, absolutely. And then another book that I just fell in love with, and actually, here Ben Rockwood again over at Chef, and another person who has been very influential in the DevOps movement. He mentioned the original TED Talk by Simon Sinek, "Start With Why."[†6] And I watched the TED Talk, and I was "I have to buy the book." And *Start with Why* is really interesting.[7] It's based on this idea of this golden circle, which is actually something that is common in biology, and basically the golden circle is...the outer circle is the "what," the middle circle is the "how," and right in the middle is the "why." To summarize what Sinek is saying, too often we start with the "what," then we try to break down the "how." Does this sound familiar to what we recently just talked about? And then possibly we get to the "why." More often than not, we don't.

The whole point is, start with why. Start with the "why you're here." And in that book, he has amazing stories of comparisons of Apple and Dell trying to get into the iPod or music thing. Dell had supply chain, everything. But their "why" was they were a computer company. And he says that Apple's

"why" was creating awesomeness. And he gives example after example, and what's interesting when I was first reading this stuff, I actually worked for Chef, and I was able to use it as kind of a marketing starter, but it was true. From head to toe, until we got to fifty employees, everybody was a sysadmin. The CEO, Jesse Robbins, was a sysadmin from Amazon. So our "why," one of the reasons Chef was so successful early on and throughout the years is the "why" was very clear. We were there as sysadmins to try to solve sysadmin problems. We didn't...nobody took the book and was like, "Here's how you're gonna build this company," but literally worked from the "why" to the "how" to the "what."

Gene: And I love that because here's another example of, to even more of an extent of *Lean Startup*, that language is now permeated into every bookshelf in modern organizations. Very helpful for us in the DevOps community. And I think there's other books that certainly touch upon this as well. *Team of Teams* by General Stanley McChrystal,[8] in some ways, this is just a great book about functional silos just having to work together to achieve very important goals. And *It's Your Ship* by Captain D. Michael Abrashoff.[9] Again, it's interesting to see how much of this is being incorporated not into business thinking but also military thinking.

John: We talked a lot about learning organizations. We talked about the theory, military examples, business books. We've also previously spent a lot of time on command and control, so I wonder if we just pause for a moment and talk about command and control. How do you define it? How do you...what's the opposite of it? And how does it relate back to this topic of learning organizations?

Gene: It's interesting that command and control often has a negative connotation, but the reality is that the processes associated with it are often

also associated with things that have created incredible greatness.

Dr. Steven Spear, who we've talked about before, he once said, "Command and control, it's a way to create a semblance of stasis in a dynamic, chaotic, and complex world." We also talked about how Alfred Sloan, who was head of General Motors for decades, he created a century of US economic dominance for the early part of the twentieth century, so I think it's important to acknowledge that these things are not inherently bad. They're actually associated with creating amazing things.

John: Yeah, whenever I'm giving a presentation about Deming or talking about Sloan, Taylor, I'll ask everybody in the audience to pull out their iPhones, and I'll say, "Oh, by the way, thank Sloan and Taylor, because the great economic wealth that was created during that period is probably why you have this phone in your hand right now."

Gene: Exactly, and I think one of the most interesting treatments of this is a book by Dr. Vijay Govindarajan and Dr. Chris Tremble, and they are both professors at the Tuck School of Business at Dartmouth. They wrote a fantastic book called *The Other Side of Innovation*.[10] And they've always studied great organizations. And they find it as multi-billion-dollar-a-year organizations who have sustained greatness over decades, and you can't do that without being great at daily operations. That means product development, customer support, supply chain management. And after reading about the MIT beer game, we all know how important supply chain management is. And the way we sustain greatness is through process. We codify our best-known methods on how to do something. We then create bureaucracies to protect those processes. We put in approvals to enforce compliance to that process, and the resilience that bureaucracies create is difficult to overstate. You can take out half the bureaucrats and the process

still survives. In fact, the process will often long outlive the reasoning of why that process exists.

So, Dr. Govindarajan and Dr. Tremble, they have to call it the performance engine. And they call the performance engine amazing. It allocates resources, it coordinates hundreds of thousands of people around the world. It preserves continuity, it sustains greatness. And everyone in the organization should be grateful that the performance engine exists. But they also acknowledge—and I have studied—how it can kill innovation, because whenever you need to do something destructive, something like DevOps, it is often the performance engine preventing that destructive innovation from happening.

And some of the artifacts are very easy to spot. Not just powerful, entrenched bureaucracies but functional organizations that treat each other like nation states, like sovereign nations. So, in a world where conditions that we operate in, whether it's a business or in the military, things are not static. Things are changing so rapidly, and we rapidly see the limitations of the performance engine. I think you can say now more than ever, we are in a very dynamic world, and we are relying on systems that are so unsafe and so complex. You could call the performance engine very much like living in the world of Safety-I, and we start to see that we need to replace that with something different, and I think Safety-II is another way to frame it. And I think what I...an interpretation of the business books are focusing on, what to replace command and control, it all says that we know we are reaching the era where command and control will not be enough, and the question is, what do we need to replace it with?

And I think the answer really is what we've been talking about in this module, which is the opposite of command and control which is dynamic learning organization. I think that, by far, the most famous example of a

learning organization is Toyota, where they integrated a scientific method of learning into everyone's daily work.

So, you look at all these authorities that we studied in this module: Dr. Peter Senge, Dr. Chris Argyris, John Shook, John Boyd, Norbert Wiener, all of these are talking about learning organizations, but of all of those, my favorite on this is Dr. Steven Spear, who we've talked about before.

John: Yeah, and I think the theme here is that we've coupled all this knowledge by this idea of dynamic learning organizations, and it just reminds me of Dr. Spear's *HBR*, *Harvard Business Review*, about Toyota, and it's a quote that has stuck with me over the years is Toyota was a community of scientists continually experimenting. So, Gene, when we talk about Dr. Spear, why do you think it's important to discuss him in this section when we're talking about learning organizations? Some people might ask, isn't he just a manufacturing person?

Gene: Yeah, and that is true. In fact, that famous *Harvard Business Review* article that you described in 1999, that was actually based on his PhD doctoral thesis that he did at the Harvard Business School. As part of that work, he actually worked on the assembly line of a Tier 1 Toyota supplier for six months. Amazingly, the Toyota executive said he could do that but only with the condition that he first worked at a Big Three automotive manufacturing plant for at least thirty days. I think the reason is that he would see many things, but without the context of what a more typical plant would look like, he wouldn't appreciate the learnings. So, over the years, he's extended his work beyond just manufacturing. He worked on studying the engine design processes at Pratt & Whitney, building a safety culture at Alcoa, to the design and operations of all nuclear reactors in the US Navy seagoing fleet, another complex system fraught with safety implications, and extending his work to healthcare.

What I love about his work is that he, more than any other researcher, has done the best work in describing what are the structure and dynamics of learning organizations. How are they set up in terms of the organization and what are the behaviors that you see in the daily work? What I love about his work—and this is all in his book *The High-Velocity Edge*—he makes this claim that while designing perfectly safe systems is likely beyond our abilities...

I'll just pause there for a moment and say there is very little work that I can think of that is more unsafe than the ones that we do every day in our daily work. So even in our work, safe systems are close to achievable when four conditions are met. What I'd like to do is just go through what those four capabilities are and describe some of the technical practices, such as architectural practices and cultural norms we see in DevOps to show how practical of an aid his work is in terms of guiding the creation of dynamic learning organizations.

The four capabilities that Spear asserts are one, we have to see problems as they occur. Two, we have to swarm to solve problems and create new knowledge. Three, we need to spread knowledge throughout the organization. And four, leaders need to create new leaders.

So, the core of capability one is that we have to see problems as they occur. I love the way that Spear phrased this. He said complex work must be managed so that problems in design are quickly revealed. We only do that through the relentless testing of assumptions. In our work, what comes to mind is assertion statements in code. In fact, it's an interesting assumption there is that if we fail in assertion, it's better to fail the program than to continue running in a state where we've drifted off the norms that we need: continuous builds, continuous testing, and the proactive monitoring of the production environment. Everything has to be radiating telemetry so that we can see how they're behaving and especially to know when things are not

operating as designed. When the system as imagined does not actually match the system of reality.

John: Yeah, and you know, what we've learned from Toyota is this idea of going to the Gemba, seeing the problem. I think it's taken time in DevOps to figure out what that means, and I think an example of that is the postmortem, which is you don't just accept we saw that yesterday. It's a virtual form of going to the Gemba. No, we didn't just see that yesterday. Let's break down this thing. We're not gonna accept. Going to the Gemba is not the call the person say, "What happened? Oh this. Okay, we saw that already." It's the, "I'm gonna come and I'm gonna look at it."

Gene: And I love in the postmortem processes at Etsy they talk about, what were the events that took place before the accident? What were our assumptions? But also overlaying that on the timeline of telemetry. Someone at Etsy famously said if we have a religion at Etsy, it is the church of graphs. If it moves, we graph it. If it doesn't move, we graph it just in case it makes a run for it. One of the Spear phrases I love is when you study high performers, we always focus on their altitude when what we really should be looking at is a rate of climb. So, as evidence of this incredible focus on metrics at Etsy in 2011, they were tracking 200,000 production metrics. By 2015, they're tracking 800,000 production metrics. It's all about making it easy for the developers to create telemetry as a part of their daily work.

John: Yeah, one of the things I think also Etsy...they didn't introduce it...but this idea of superposing or transposing other information on a graph. Not only how often you deploy, but this is when we deployed, and here is what happened from the monitoring of the CPU or systems. To be able to see those multidimensional and put things in context...That's telemetry versus monitoring.

Gene: I love how very vividly they describe how telemetry enables safe system work and how it enables safe deployments. People don't go home until they make sure that whatever they deployed in production is operating as designed, has had no other horrendous consequences that were unforeseen.

John: We talked in an earlier section about safety and John's thesis. One of his conclusions in his thesis was very simple and obvious that he had heuristics. The first heuristic was basically make the assumption that it was the last change. So, there again, that telemetry of on the screen, everybody can see it, something went wrong, let's go back to that last red line that was the deployment.

Gene: And then what better way than to map to capability one, which is all about see problems as they occur. The last known change. I love it.

So the second capability in Spear's model is that we must swarm problems and solve them in a very specific way. In other words, through swarming, and not just to fix problems quickly, but to build new knowledge. One of the passages I love from *The DevOps Handbook* is what is the opposite of swarming. It was observed that at that General Motors Fremont plant that we've talked about so much, the place of the birth of NUMMI, there were no effective procedures in place at tech problems, nor were there explicit procedures on what to do when problems were found. As a result of this, there were instances of engines being put in backwards, cars missing steering wheels or tires, and cars having to be towed off this assembly line because they wouldn't start.

So, the part of capability two is to create as much feedback in our system from as many areas of the system as possible sooner, faster, and cheaper, with as much clarity between cause and effect. The reason is that the more

assumptions we can validate, the more we learn. The more we learn, the more we can win in the marketplace.

John: A DevOps example of a swarm capability too is breaking the build. They break the build. In fact, there are many organizations that will actually make fun of the break the build...not to be mean, but to emphasize the point that this is a swarm opportunity.

In fact, one of my favorite stories...I was visiting a credit card company once, and they had this DevOps war room, and I noticed there were those little Nerf bullets all around. I just asked, "Oh, do you get frustrated and just have a team Nerf war?" No, they showed a little Gatling gun Nerf gun, and it was set up on a raspberry PI. When the build broke, they geopositioned it to the person who broke it, and it just started firing away. Anyway, but it's a fun way to create this swarm notion of, "Hey, the build is broken. We're all working for the same master. We need to figure out how to get this moved."

Gene: I think what that illustrates so well is this cultural norm that when a build breaks, there's no more important work than to get that system back into a running state. It doesn't mean that we have to drop all our work and everyone fixes the build, but if an engineer who is just pegged with a Nerf bullet who needs help from somebody, well then, there's no work more important than helping that engineer get that build into a green state. Similarly, with when somebody breaks an automated test, there's no work more important than to ensure that we get back into a safe, deployable state. I think what's interesting to me is that it even extends beyond the deployment pipeline, even to doing peer reviews.

Elizabeth Hendrickson—someone who's had a tremendous influence on me—she'll pioneer so many of the practices around automated testing. She said when someone...If I'm working on whatever I think is the most important thing it is in the moment, and someone asked me, "Gene, can you

look at my changes so I can push into production. I need a peer review, a plus one, plus two," I should drop whatever I'm doing and help that person because we know the longer that person has to wait to get into production, the worse the outcomes.

And so, it really says that swarming doesn't have to mean drop everything. It really means a prioritization of getting into production over whatever we think is the most important, and tomorrow it could be the other way around. I need the plus one or plus two.

I think one of the best examples...We all love the statistic of how many Andon cord pulls there are in a typical Toyota plant. It's 3,500 times a day. And it's always shocking. We have to pause a moment here to say when the Andon cord is pulled, the entire assembly doesn't stop. In general, you've got fifty-five seconds or whatever the Takt time is to resolve the issue, and even then, you can actually chase the work down the line with the missing bolts or whatever. But the question is, why would you do it so frequently? We've talked about how important it is to introduce tension into the system because every Andon event represents a learning event.

And I think the best example of capability, too, for our world that shocks and awes people is Google. They had 15,000 engineers back in 2013. They're working on 4,000 simultaneous projects. Everyone's working out of one source code repo, one CI system. Back then, they were doing 5,500 code permits a day. They're running 75 million test cases daily. The question is, why would they do that? Running tests consume electricity. It generates heat. You have to always groom the test suites to run quickly enough to give a fast feedback to developers. And in *The DevOps Handbook*, one of my favorite quotes is from Eran Messeri. He said, "It is only through automated testing that can transform fear into boredom." And I just love that quote.

John: We've talked about this before. If we go back to Toyota and why they did the Andon cord and there's the famous Kentucky plant story where they were asked how do you create I think it was twenty-two cars a day, and the answer was we pulled the Andon cord 5,000 times a day. And to me, I've watched that number track at Google, back I think in '13, it was 75 million. I think 2016 they came out again and said, "By the way, it's 150 million automated" And when I'm presenting that, I want to say that it's another, I want to say it's twice...Can you get the gravity of 150 million automated tests a day? And I say, that's that Kentucky plant.

When you create that kind of resilience, you get that kind of speed. You're creating tighter tolerance of feedback loops to a scale that is unparalleled anywhere.

Gene: And to sum up, that's all about capability too. Swarming doesn't just fix problems faster, but it is the ability for us to create learnings in our daily work. Capability three is probably one of my favorites, and John, you and I know when we were working on *The DevOps Handbook*, this is where I felt like we had the biggest blind spot. And in some ways, there's a narrative that said, "That probably introduced a two-year delay in *The DevOps Handbook*." And that was revealed when I took Spears's workshop at MIT. And capability three reasserts you have to have some mechanism to spread new knowledge throughout the organization. In other words, how do you translate and convert local discoveries into global greatness?

I think what's so interesting is that in our world, if you ask Randy Shoup from Google, he said, "If you asked anyone at Google and asked them what is the most powerful preventative mechanism to ensure safe systems...And the most common answer would be the single shared source code repository." In other words, whenever you want to solve a new problem, you

first look in the repo because somewhere along those 30,000 engineers is probably someone who tried to solve that problem before.

In fact, there is this amazing tradition that there's only one version of each library allowed at Google. And there's actually a library owner who's responsible for migrating everyone from Version N to N+1. And I think for operations and security, this is so powerful because we don't want knowledge in people's heads. We want them in the code so that anyone can pull from that repo and automatically inherit the best-known understandings of how to solve a problem or how to secure a system.

John: So, it was interesting, I visit a lot of companies. It's one of the things I love about, if I have to call it a job. I get to visit these companies and spend some time with them, and literally last week I was at the e-manufacturing, commerce, and there was a team. It was a large company, and they were actually building, serverless, a whole new payment system on Microsoft's functions are serverless. Completely head-to-toe. And I asked them some questions about, "How are you integrating with the cloud services people?" And they quite honestly said, "There's some tension because we're so far ahead of them." And then the director of that team said, "Since you've asked, how would you recommend us to create scalable learning?" There's a good example of we've gone so far out, even the cloud people are behind us. You know, the classic cloud delivery...

And I had to step back. I'm like, now I've gotta tell a team that's moving so far ahead of production, a Fortune 500 company running a serverless head-to-toe payment system, PCI, all that stuff. And I step back, and I said, "Well, traditionally, the way we have solved this problem is...I'll give you the easy answer first, is the hackathons." And we've talked about this in the DevOps community. A lot of companies have spent a good amount of time either creating a reasonable cadence, maybe some very aggressive, once a

week, some once a month. We know that Capital One aggregates thousands of people together for a development jam.

And then I'm like, "Okay, but I told you that was the easy answer." And then the light bulb went off, "Well, the harder answer for you to...Hackathon says you could probably kick that off once a quarter. But I think the Dojo." So again, how do you...how do we create these cross...If I hadn't made the point, the problem he was trying to ask was, "We have this imbalance with different teams learning different skills, different technologies. How do you scale that across?"

I mean, Toyota had a very good model. Mike Rother talked about it in *Toyota Kata*, how they were maniacal about when they learn something, they had a system for spreading things out. And here's what [the e-commerce company] told me, and he's like, "I don't know how." And I said, "Well, certainly hackathons. But the Dojo."

The Dojo...the thing I love most about the DevOps Dojo...Ross Clanton had mimicked it at Target. Ross Clanton is over at Verizon and doing it again. It's this idea of an immersive learning place where different groups can come in, and you're just constantly aggregating not only the learning skills but the technology.

One of the things I loved most about the Dojo, as I describe it to people, is a pull model, not a push model. Because sometimes when we're trying to get other groups to adopt our learnings...I want you to become a learning organization. In large organizations, we're constantly going and being evangelists and "You should try this" We come back, "Nah, you gotta try it." Dojo creates a pull model where people want to come in and they actually eventually queue up and have to wait in line to come in.

Gene: In fact, we document this in *The DevOps Handbook*. I remember shadowing Heather Mickman at Target and just going to visit the Dojo and

the gratitude of these teams who were in there. They were saying, "We've known about these ideas for years, but we just never had the time, nor did we have the experts on staff." And to be able to come into this Dojo for thirty days and have four people assigned to help us, that was just...It was probably one of the most incredible parts of that visit. So that's absolutely capability three. And other examples I would throw in there are learning days, internal technology conferences, and even blameless postmortems. One thing I would add is that notion of publicly broadcasting postmortems to the rest of the organization is such a powerful mechanism to elevate all our skills.

John: And we've been fortunate enough that there's quite a few companies now that have run internal DevOps days. I know you've been invited to a few to speak. And they're really interesting because they're a mixture of external speakers and internal speakers. So, you might have a DCO or some executive level kick the thing off about, "Hey, everybody. Here's what we're trying to do here. I'm gonna introduce now Gene Kim." And then the next person is someone who's leading a large implementation of DevOps. And then there's the open space. So, actually, organization's another tool for spreading that knowledge that we see work very effective in the DevOps movement is this idea of creating your own internal DevOps days.

Gene: Absolutely, and I think there's one last thing I want to mention that is surprising, is functional orientation is also a great way to help with capability three. In other words, you have all the DBAs in one place, and all the Devs in one place, and all the QA people in another place. And it's very different than the cross-functional team that we often associate with DevOps. But at Google, all 1,300 site reliability engineers all report to Ben Trainer, the VP of SRE. And part of that is so they can control the...have a uniform hiring process, they can ensure a uniformity of culture and skills, and they would be matrixed into the product team. So, a functional

orientation is also what they noticed at Toyota. In fact, they called it the second Toyota paradox.

John: Well, to keep going here, because obviously we are pretty excited about this capability, if you will. I just recently visited Facebook and Pedro Canahuati, who we talked about in *The DevOps Handbook*, and while Ben Trainer was building SRE over at Google, at the same time Pedro was doing the same thing. Now, they don't call it SRE, they just call it production engineering there [Facebook]. But one of the interesting things...a lot of those traits of this group is to share responsibility, the site reliability engineering expertise, but one of the things I love what they talked about is...At one point, they learned that they actually had to make that a pure product group under the VP of engineering.

So, because no matter what you get around it, it was still this tension of "Well, aren't you supposed to do that? Aren't you the team that does that for us?" And one of his learnings over the years was to almost move down his authority to report to VP of software engineering, such that now he was a peer group to every other product in service, so that you couldn't play that "You guys know. Well, we planned. Your service reports to my service," and I just thought it was a brilliant. Again, if you get to watch some of Pedro Canahuati's presentations, you get to see another variant of this thing, SRE, even though they call it production engineering.

Gene: Capability four is that leaders have a different role than in the typical command and control organization. Say, lead by developing, leaders create new leaders. I love this quote that actually came from one of the CIOs at Target. He said, "My goal is not to direct and control, but it is to guide and enable." And I think one of the best verbalizations of this is in the DevOps community is transformational leadership. Dr. Steve Mayner—he did his PhD on it, he's spoken at DevOps Enterprise. And when I asked him what

"transformational leadership" was, my jaw dropped, because he described in clinical precision all the attributes I had seen in people driving DevOps transformations in large, complex organizations.

And so, here are the five capabilities: the first one is vision. Does the leader know what the vision of the organization is? And then can that leader get in front of it? Can they help with the achievement of it, not just for relevance but to help with the achievement of the most important goals of the organization?

The second one is intellectual stimulation. In other words, does the leader challenge status quo? Can they challenge basic assumptions of how we do work? So, if you're trapped in the performance engine, you are looking at processes that have been around for years, maybe even decades. And so, to what extent are they able to effectively challenge, effectively recognize that and change those processes?

Three is inspirational communication. I love this one because it says, "To what extent can the leader inspire pride in being a part of the work?" To what extent can they overcome maybe very powerful senses of fear and uncertainty that you know. What does this new whole DevOps thing do to our career and our career path? I've invested decades to get where I am; this doesn't sound very good to me. It's not just within the team, but it also speaks to outside the team. Can they create a coalition that is powerful enough to challenge these very powerful and very common systems?

The last two are supportive leadership and personal recognition. And so, those two are very commonly associated with servant leadership, and that's at the front and center of Agile. But what's interesting to me is...maybe to oversimplify, the difference between transformational leadership and servant leadership is that servant leaders focus on the needs of the followers, whereas transformational leaders also focus on the biggest needs of the organization.

John: Yeah, I've always thought of that same kind of notion, but David Marquet wrote this, the intention-based leadership. It's more of a...instead of a top-down, it's a bottom-up thinking about how you become the person who helps. You are the one who gets them out of a jam, but I also think that this ties really well into something I think a lot about, which is...I just used the true north, because it comes from Rother, but...Look at certain companies that in order to be a vision, the company has to have a clarity of the vision, which Rother describes very well in *Toyota Kata* as the true north, so I've had this opportunity...In fact I talked to Mike Rother in Detroit about this, and he thought it was interesting. If you go to SpaceX, you'll see this mantra of anything you've asked to spend money or any time you want to hire somebody, there seems to be this pushback question: "Does it get us to Mars fast? Does it get a person to Mars fast?"

Chef Con 2016, Alaska Air gave this presentation about time. And they give this great example: as the plane pulls up to the gate, and they actually have a timer that if they can get the door to be opened in under sixty seconds, they see that as buying two new 737s.

But my favorite recent story is visiting Fannie Mae. I'd been spending some time on security, and DevOps and DevSecOps, and so one of the women that run this ab sector...so Fannie Mae will tell you, "We put people in houses." On the front page, it's about the experience of our customers and getting a loan as efficiently as possible; this is trust, and it's seemingly a growing true north there. When that vulnerability in 2017, the struts to mid-March...What happened to Fannie Mae is that, maybe on a Wednesday—I might not have the days exactly right—but on a Wednesday, the vulnerability was exposed. And it was actually parsing routine, but too deep.

The patch was out two days later, but it was a longer time to get any distinguisher files. And the women who ran ab sec knew that you could fix with the patch any moving forward ideas going through that the delivery

knew. But she knew there were just tons of code that had that vulnerability, and things like WebSphere, and different parts of all the infrastructure.

So, on a Friday afternoon, she went to the senior management and said, "I believe we have to take the production loan application down." And they...Imagine that on a weekend, too, which is you know, probably a very active time for house loans. And the question that she proposed is: Look at our website. Are we really doing this if my suspicion is hundreds of millions of customer's roaming data will be out in the wild. Do you live up to that vision of what kind of organization you want? So, there is that capability four but there's this test of an organization where you say where the...and again, they literally shut down the loan application. She worked all weekend, she wrote a little script to hit all the internal URIs to find out where that parsing module was, and by Sunday night, they were able to mitigate that parsing algorithm and they were up Sunday night.

Gene: We recently had the opportunity to watch a very distinguished panel, and Spear said something I'll never forget. There is this wonderful symmetry around psychological safety, and you had mentioned this earlier. It's not just empowerment of the line worker, but it also requires a stunning level of humility from the leader. And you're adding this other notion that is so much reinforced by a strength of vision, strong enough that can actually guide the right actions.

John: Yeah, and I think the danger that none of this is platitude, in spirit, but we run a danger...I just listened to this great audio series. We need to have a vision, but vision without a backing or… You know, like the Fannie Mae is just a walk-to-walk story. Let's shut this down because what we're about to happen will not match what's on our front webpage.

Gene: So, John, you had asked, "Why Steve Spear? Isn't he the manufacturing person?" And in short, I would summarize by saying yes. He came from manufacturing but has extended it to many, many different domains, very highly complex, very safety-related. You know what I love about the four capabilities is that it illuminates practices that I find really exciting. I think just even our conversations, we get so excited about it, and I love it because the capabilities explain the causal model of how these four things put together lead to the outcomes we want, whether it's learning, creating a dynamic new organization, or creating safety, and ultimately about the mission of the organization increasing, helping solve societal problems.

John, I recognize that we have just made some incredible claims. We are suggesting that it is all about learning, whether it's from Goldratt to Deming to Lean, to safety culture and now to learning organizations. It really is about learning, isn't it? It seems like we see it everywhere, whether it's a set of DevOps Days or DevOps Enterprise conference. We see learning happening everywhere we are doing work.

John: Yeah, you know I always go back to...Andrew Shafer gave a presentation at Velocity 2013 and it was called "There Is No Talent Shortage"[11] and I know Andrew well, and he was...There was a shock factor where we were all sitting in the front row. "What do you mean there is no...We can't hire anybody. There is a talent war going on in Silicon Valley." And the head fake was you need to become a learning organization. And it was the first time I really saw somebody in earnest express Senge and all these things come into our space. It's where his famous quote came from: "You're either the learning organization or you're losing to one."

I think that as leaders in this community, this thing we call DevOps, we have to just continually create that momentum. "I've learned from you,

you've learned from me, I've learned from Ben Rockwood." And then list Andrew and the presentations, and we watch people bring in presentations with books, and we say, "Oh, I should go look at that," and I'll read that. And we just keep pushing it towards the middle of this thing we call DevOps.

Gene: Yeah, I couldn't agree more, in fact another way to interpret transformational leadership is that the role of the leader is to create the conditions where learning is not only possible but valued and viewed as a necessary part of our work.

John: Yeah, and you know, I think there is a really good example at the DevOps Detroit where we both had the opportunity to visit...There was a presentation there, it was called "The Psychology of Estimation: Faster Is Not Always Better" by a gentleman named Amin Yazdani,[12] and it both hit us, but you...

Gene: You're sitting in the front of the room, I was sitting in the back of the room with Mike Rother. That was such a phenomenal presentation because he brought up the book *Thinking, Fast and Slow* by Kahneman and Tversky,[13] which I read before, but I loved his presentation because it was a very practical application and explained the cognitive biases we bring into task estimation that almost guarantee that we are going to get the outcome wrong. And so, it wasn't just sharing a book, it was sharing his learnings and viewing a problem we have all faced in a different lens. I thought it was a fabulous presentation.

John: Yeah, and I think one of the questions is...I think we've all been excited in this DevOps move right from the get-go. We've felt part of something, and you know, the question I think we ask ourselves is, "What's different about this?" And a big one is that it's a great example of not only a

collaborative effort of people who want to share ideas, but that people then come back with those ideas and then teach their applied learning of those ideas. That was a great example of this gentleman that's not only read *Thinking, Fast and Slow* and some of the work by Kahneman and Tversky, but then gave us an example of how he applied it in his daily work.

Gene: You know, I've always loved studying the bookshelves of technology leaders, and there is a certain commonality, but my conclusion is that we're all trying to teach each other and teach ourselves and stitch together what we need to learn to achieve our goals. And so, what I think I love so much about the DevOps community is that fundamentally, we are a community of learners.

John: Yeah, no, absolutely. And I'll just go back to probably one of the earliest definitions of DevOps, and still today probably my favorite is by Adam Jacob, the founder of Chef. And he said DevOps is just a cultural and professional movement and emphasized full stop. Like that—we're done. And the only thing I would add is that this cultural and professional movement includes this voracious appetite to be learning and learning organizations.

[†]Simon Sinek's book is titled Start With Why. His TedTalk is titled "How Great Leaders Inspire Action."

Module 7

Lean, Safety and Learning From the Experts

This module features a transcript for the DevOps Enterprise Summit 2017 panel discussion "Convergence of Safety Culture and Lean: Lessons from the Leaders" between Sidney Dekker, Steven Spear, and Richard Cook.[1]

Gene Kim: John, we previously covered Lean, safety culture, and then dynamic learning organizations. We covered works from Dr. Sidney Dekker, Dr. Richard Cook, and Dr. Steven Spear. You had this idea to bring these people together and explore where there was commonality and where there were places of genuine divergence. In 2017, we brought them together and some remarkable things happened. John, how did this all come to be?

John Willis: Yes, so, we talk about in this work of how you introduced me to Goldratt, and then you actually introduced Mike Rother's book *Toyota Kata*, and Dr. Spear's book *High-Velocity* [*Edge*]. So I started becoming a deeper student of Lean in a certain way, and then I had also been reading Dekker

and a lot of Dr. Cook's work, and I had the opportunity in 2014 to meet Dr. Woods and Dr. Cook in person. It was at Velocity in New York City. So, I figured I'd approach them about...It seems to me there is some commonality, and it was shut down pretty quick...

Gene: Commonality with Lean.

John: Lean, I'm sorry, yeah, Lean, and they would say what they were doing, and I wasn't going to take no for an answer on this. That's when I started having conversation with you about this idea that if we could get these two groups together and force them to have this discussion, and we had this opportunity in DevOps Enterprise Summit 2017 in San Francisco. To actually spend four hours with these people, and creating that, to getting to a place, and the end result was a panel, a thirty-minute panel, where, Gene, you moderated, and Dr. Cook, and Dr. Dekker, and Dr. Spear were all in the panel. It was great because that was the point, after four hours, our hypothesis, I contend, was correct.

Gene: Awesome, and I think our hypothesis was that we have so much to learn from both, and they are both mutually reinforcing movements. So let's take a listen to what the panel had to say.

• • •

Gene: Good afternoon, everybody. Why don't we bring our panel out. Dr. Cook, Dr. Dekker, and Dr. Spear. I want you getting seated. Let me share with you what we did today. We spent four hours together this afternoon to really explore safety culture. Where they intersect. Where there might be disagreements. I have to tell you, it was one of the most professionally rewarding things I have done in a very long time. It was a magical, and if I may be honest, very stressful experience. To be able to do this with John

Willis as the co-pilot was just fantastic, and I learned so much. I think it lives up to the notion that I had said once, which was that I think it will be historic, and I certainly believe that it will be important.

One of the things I found so meaningful is that they actually give me a lot of concrete advice in terms of how we can accelerate the DevOps movement, how we can take advantage of this opportunity that we have, and how we can increase the likelihood of this movement succeeding. So we will make the video available to share with you and a much broader audience sometime in the future.

So, I have prepared three questions, and hopefully you'll get a taste of why this is so rewarding to me. Let me start with, first, Sidney Dekker, Richard Cook. Both of you have had time to spend in the DevOps community. What are the key places where safety practices can help us, Sidney, Richard?

For the last four hours, we couldn't keep them from talking, and now they are not saying anything.

Sidney: It's called stage fright. All these people, Oh, my God. I think a qualification is in place. Perhaps there are none. I think Richard might speak to that, but if there would be...We've learned through lots of damage and dead bodies that safety is a culture that allows the boss to hear bad news. There [are] two problems with that. One is to give news to the boss to begin with, and for the boss to be open to that news, and for the boss herself or himself to be able to share bad news. But then the question is, what is bad news? What constitutes interesting feedback that you really think you need to be concerned about?

It's that discussion that I would like to impart. Other than that, and this is a discussion that we certainly had, Gene, is the issue [that we] don't see people as the problem to control. Don't think that your people need

controlling through more procedures, and more training, and more guidance. No, they are your problem solvers. They know what to do; don't tell them what to do. Ask them what they need to be better at it and then don't sweat counting negatives. Understand how success is created.

Gene: You said something that I found very significant, which is not just leaders. It involves leaders and people on the front lines. That there is a symmetry.

Sidney: Particularly in some other cultures. It's not as problematic as in North America, actually, but it's easy to think that we just always have to legitimate the voice from below who doesn't have status. Oh, make them speak, and make them speak up, and make them feel safe. Which is incredibly difficult to do, and by the way, I think none of it sits on a good example of an organization that does this perfectly, at all, by the way. But it is also the leader, she or he needs to be able to say, "I don't know, I need help. I am actually concerned about this. I need you guys, you people, to help me do this and figure this out." And sometimes it's quite difficult for a leader to be seen to be losing faith, to not know the answer, until that culture of psychological safety, if that's the term that you want to use, goes both ways.

Gene: Awesome. Richard...

Richard: I think one of the problems that we face, particularly in high-velocity production environments, is that we are better at forgetting that we are at learning. We're pretty good at learning, but we also forget very fast. We have a forgetting problem that's probably more significant than our learning problem in many settings, and our forgetting is so quick because we're moving on to other things so fast that we can essentially lose sight of the lessons that we learn. We need to really work at trying to sustain the lessons learned and the experience over time, and that means not simply dispensing

with the problem and considering that its done. John Allspaw talked to this a little bit yesterday when he talked about the fact that people in their PMs were saying we all know what happened, let's just fix it and move on. But I think that we really need to be able to remember these events, and if you understand the Heinlein phrase, grok them in fullness overtime, because I really think that's crucial.

Gene: Actually, before I ask you the question for the Lean community, you actually said something that was very memorable to me about the role of the leader. Often… the stunning humility of the leader. Could you repeat that so that everyone could benefit from that observation?

Sidney: Have we said profound things? I don't recall this at all. Well, it's on video, so you can dig it out somewhere.

Gene: You remember, Steve?

Steve: Yeah, so it picks up what Sidney was saying, which is the adversary for all human activity is ignorance. If we had perfect knowledge, nothing would ever go wrong. The reason things go wrong is because we didn't know what really we should have been doing or really how we should have been doing it. So, if we view ignorance as the adversary for all our well-intentioned undertakings, then what we have to do is make sure that we behave in such a way that we recognize we're ignorant sooner than later.

Sidney: Which requires humility.

Steve: Which requires humility to say, "I'm in this situation, and there are probably some fairly important profound things where I'm actually quite moronic about these things." And to Sidney's point, I think he's right on it. It has become a very convenient cliché to say we'll create an environment of

psychological or emotional safety, we'll give voice to the person on the deck plate, the shop floor, the bedside. We'll empower that person, but then the senior leader still acts like a pompous jackass. So really, how well are they coaching and modeling exactly the behavior there espousing?

Gene: Any comments to that? Otherwise, I will ask Steve the second question.

So, Steve, the corresponding question to you would be: What would you consider the important Lean practices that are important for DevOps? Is Lean good for us, and what are the right things to really learn from the whole Lean movement?

Steve: So, we had a longer conversation about the source of the Lean movement, and it is important to understand that the headwaters for that was Toyota. What was the start point for Toyota? It was incompetency in manufacturing cars that would be competitive in the world market, and this Toyota production system orbited around the notion that what we do, we're really, really poor at, and if anyone catches on that were really poor, we'll quickly become irrelevant. And so you had Toyota leadership from the very beginning managing their systems with this idea of what we have to do is elevate and reveal what's wrong so we can come to a better understanding of what we're doing, and why we're doing it, and how we're doing it. So anyways, what happens with Lean is—and this is a longer story—but what happens with Lean is that Toyota's approach towards managing to identify ignorance and convert it into useful knowledge becomes a set of tools by which you can have stability over a system which would otherwise be chaotic.

But of course all stability is just temporary because the things you are doing are constantly changing in an environment which is constantly changing. So a direct answer to your question of what's the most important

thing we can extract from, and I'll say Toyota, not necessarily Lean, because that term has so much meaning, the most important principle is that of the Andon cord. If you think about what the Andon cord represents, it's the coupling of standard work, which is the agreement you and I have that this is, in the moment, our best-known approach. Not the best approach, but our best-known approach for accomplishing something. But it also is coupled with this humility, this recognition that our best-known approach will be inadequate. We don't know when; otherwise, we'd have a better-known approach, but it will be inadequate. And because it will be inadequate, we have to have a mechanism to calling out it's inadequacy the moment that first inadequacy is recognized. So the Andon cord.

Gene: And you feel like that's an embodiment. Actually, just to think out loud for a second, my first reaction is uh-oh. This is uncharted territory. We didn't actually quite get here. We thought this might be a trigger word in the safety community because Andon cord means work stoppage, and cease all operations, which could be considered an anti-pattern.

Steve: Well, can I just elaborate on that? I think that the literature which attempted to explain Toyota, or at least describe Toyota, said, "Oh, the Andon cord. If someone just has a problem, they just pull this cord, and everything comes to a screeching halt. When in fact, that's logically undesirable and also not what happens. What the Andon cord does is it declares that there's a situation occurring which was neither expected nor desirable, and it has to be swarmed and investigated in the moment. Now, that doesn't mean that all these other situations which are actually working stop; that would actually not make sense. It's just that that one right there deserves attention. And just a quick reference to the medical community—if you think about all the instrumentation that goes on a patient, particularly one who's in critical condition, the reason that's there is to indicate, oh,

there's something going on in the moment which requires attention. Now, you don't shut down the hospital just because the person has an irregular heartbeat. You know, you deal with that one, and let everything else continue till you discover an adjacent abnormality.

Gene: Sidney, Richard, can you react to that?

Sidney: Yeah, one of the things that I think has made other industries, and I don't want to be imperialist in any way about that because no one has got this figured out perfectly. But it's to then tell stories about that irregular heartbeat. Right? How could we not have foreseen that happening? And so if you're willing and honest and open to tell those stories and share those stories, not only—and this is certainly something that we talked about and that I think is critical—not only is it people love to tell stories and we do this naturally. We're geared to remembering stories and telling stories even before. We don't inhabit a universe of concepts and numbers. Well, he might because he's MIT.

Gene: How many times did MIT come up in the four hours? That makes, like, six hundred.

Sidney: But other people inhabit a world of stories, and this is what we'll remember. And one of the things that Richard threw out there is—no, sorry, that Steve threw out there as a challenge was, how do we keep sensitizing the DevOps community to the risk of getting it wrong? Because the stuff you guys are playing with actually matters. It doesn't seem to matter when you're sipping your Coke and you go, "Oh, twenty-two hours, I'm still good. Let me bang some other code." If that's what you call it. Probably not. But there are consequences to getting it wrong.

Now, you cannot generically sensitize people to the consequences of getting it wrong, as in a little poster on the wall. Oh, forget that this is risky stuff, or whatever. I mean that doesn't work. However, telling stories about how things went wrong or almost went wrong, that is remembered. We were having dinner last night, Richard was present as well, and so we actually, and so he and I got plenty of war stories to share. Because something is either good or a good story. All right? And so, but we didn't. The war stories that were told came from the DevOps community.

And it does something else. It not only builds his memory trace of "Oh, hang on, I've seen this." This recognition primes action. That way you go, "Oh, I need to do this because the other guy had this in that story. I remember that." There is more than that. Out of it arises this ethic of what it means to do right, to do wrong. Of what it means to be a good professional, to be a good practitioner. And it's that trace that doesn't exist yet, as Richard's beautifully identified, because you are such a young profession, all right?

Gene: In fact, Richard, I had asked, I had the opportunity to ask each one of these scholars for advice. What advice would he give us as a community to help us achieve our goals? One of them, I mean, could you actually—you can either say city coverage is fine, or do you want to expound upon that advice? Because it actually, I found it very meaningful, and I realized that I have actually made a mistake in terms of how we...Did you want to add on to that?

Richard: I think we are, the work that you do has lots of circumstances where you can't simply say I'm just going to close up the laptop and go home. You have to solve the problem. You're confronting it; the situation is getting worse. You're in a situation of escalating consequences. You have undesirable options only. All of the choices are bad ones, and you're

choosing between bad options. There's an old surgical saying, if there are more than two operations for any condition, none of them are any good.

Gene and Sidney: [*laugh*]

Richard: And I'm afraid that you very often are in that situation. We respect the fact that you have to make decisions under those circumstances, and we would like you to be as empowered and knowledgeable and supported in those circumstances as possible so that you can make those decisions. We understand that those decisions may not always turn out well, but you're the one very often who has that burden and that responsibility. And like the pilot in the aircraft or the surgeon in the operating room, we expect whatever the long-term value for, of learning from the experience might be, we expect you to act at that moment in the most judicious, thoughtful, and prudent way that you can.

When we look at you, when we watch you and what you do, especially those of you who've got a few gray hairs or who have gotten a little bit of experience, we notice that you are very careful about trying to help other people who don't have that level of experience learn how to approach these problems thoughtfully and taking into account the ways that things can turn out. We think that's probably pretty much crucial to what DevOps is. DevOps is not simply the practice of fixing problems or generating velocity. DevOps is also the practice of building a community of people who do DevOps. You are the only sources of information about how to do this stuff that are available to the people who are learning how. And so, we recognize that you have a certain responsibility by virtue of your knowledge, your position, your status, to take on the care and feeding of those young people as they are learning how to do this sort of work.

Sidney: Yeah. Me too.

Richard: We think that would be a really important.

Sidney: Mentor, mentee. Yeah.

Richard: Yeah.

Gene: And, in fact, I misattributed where the advice came from. Because it actually started out with Steve saying the opportunity may not be in the reward but in the consequence. And my reaction was wait a minute. Aren't you going to freeze everybody into paralysis because we scared the crap out of everybody because of what could go wrong? Could you expound upon that?

Steve: Yeah, echoing what these guys have been saying, what you all do has consequence. Whether you're writing code for physical systems which are far removed from you and people depend on them for their wellbeing and their safety, or you're writing code for communication or financial systems where people are dependent on those to function well, it's worth thinking about, what's the consequence of getting it wrong? Because if you get it wrong, there's someone downrange who's going to suffer for it going wrong. And I want to tie into Sidney's point about telling stories. We worked a number of years ago with health care providers in Pittsburgh, and this one fellow, Rick Shannon, who is head of critical care there, was trying to motivate people to worry about complications. Central line infections, ventilator pneumonia, that kind of thing. He kept showing them data, and everyone had an excuse for the data. Our patients are sicker, they always are, these are sick people to begin with, we're above average, etc. and he made no persuasive progress that way. One day, people came in, and in the breakroom were posters of patients. Normally when you go into a hospital and you see posters, it's the patient you dealt with, the mother who gave birth to a healthy baby, the person who regained mobility, and when people

started asking who these people are, Rick started explaining, "This is the guy, because of a complication, he'll never have a catch with his son on a Sunday morning when the Steelers are playing in the afternoon. This is a woman whose daughter will never know her because she passed too early because of a complication."

So it was these stories of what the risks look like that prove to be, in the end, highly motivating.

Gene: And my claim was, "God, Steve, all you're going to do is scare the crap out of people, you're going to paralyze people," and you said something somewhat startling.

Steve: Well, fear doesn't necessarily have to lead to cowering. Cowering only occurs when your brain almost literally short circuits and gets into a do loop of "What do I do? What do I do? I don't know, I'll just sit on the floor behind a chair and hope the problem goes away." But that's not what we're talking about. What we're talking about is with discipline, rigor, energy, enthusiasm, and optimism recognizing moments where we're not understanding what we should be doing and how we should be doing it, saying, "Yeah, but I know how to address that, because I'm trained and practiced in part of a profession of problem solvers," so it need not lead necessarily towards paralysis.

Gene: You know, just to reflect a little bit, this distinguished panel was giving this advice to John Willis and myself, I realized I had made a mistake. I mean, in the experiences that are shared, I think we go out of our way to pick the very successful ones. These are the triumphant people who create a coalition, they mobilize and create powerful incumbent systems, they triumphed. And it made me realize that there's equal value in covering the other side, and if you can, help me make that more concrete. What is the

value of leaders that have captioned these stories of not necessarily failures but of near misses?

Richard: It won't happen in here because of the social constraints on the way that we talk about things in this sort of thing, but after you go to the hotel, and you have a couple of beers and sit down and talk with other people, the stories are not about the triumphs, okay. It's not people standing up and talking about, "We were able to cut our costs in this way," or "We were able to do this, that, and the other thing." The stories are all about the catastrophes.

Steve: Or near catastrophes.

Richard: Or near catastrophes.

Steve: Yeah.

Richard: There's another old surgical saying, which is "Good results come from experience, and experience comes from bad results." And there's a lot of truth in that, and you all know that. Sharing those stories is actually very helpful. Being able to talk about what that experience is like and what the kind of dilemmas and problems that you've been in is actually very helpful. It's what John Allspaw and a bunch of other people have been trying to look at in more detail so that we can understand something about how those things evolve.

But we have to be honest about the extent to which we find ourselves in circumstances where we don't know what to do. That's what problem solving is all about. Problem solving is what you do when you don't know what to do. And that occurs much more frequently than most people are willing to admit. I think we should be raising that up, because your skill is partly in being able to steer the system and direct it to where you want it to go, but

your skill is also being able to look at a system that's not performing well, or that's having some difficulty and figure out why, and make those sorts of corrections.

We need to understand both those kinds of activities, and we get to them by sharing both success stories and stories about the catastrophes or near catastrophes that have occurred.

Gene: And it occurs to me that this also helps to reinforce the notion of the humility of the leader. The leader is the only one who can influence the system as a whole, more than the person on the frontline. Any other thoughts on that?

Steve: Yeah, I want to tie back to Sidney's point about the leader and the modeling, the coaching, and the social rewards. Many of you know the Navy had a series of mishaps in the Pacific this last year, three collisions and a grounding, and you start thinking through all the reasons that happened. It led to a loss-of-life injury and tremendous material harm. Let's think about the grounding one. What happens, there is a crew finds themselves in circumstances beyond their control, the tide has changed, there's wind that they hadn't anticipated, the weather has gone up, they're on a schedule which they hadn't exactly planned for.

Sidney: They're driving a vessel that is made to not be seen by others.

Steve: There's that, that's right. One destroyer captain described to me, "Just to understand the circumstance, imagine going to Walmart the day after Thanksgiving, in their parking lot, when it's still dark out."

Sidney: In a black car.

Steve: In a black car with your headlights off.

But you start thinking about the issues of stories and leadership. So, let's back off from the ship that didn't actually run aground and encountered circumstances like that. So, what's the story that crew tells? Well, the story that crew tells is how they mastered the situation. They say, "Hey, Gene, man, I tell you, the waves were up to here, and the wind was going like this," and on and on and on, "but we conquered it." Well, that drives in the direction that success is through heroism. Well, they may have conquered it only because they got dumb luck, because the wind was just slightly off from the crew that actually ran aground.

The question is, do they come in and tell the white-knuckle story, which is "Oh, man, Gene, it was that damn close, and we did our best, but we were clearly not in control of the situation, and but for the grace of God go I, we might've been the one that ran aground." So what drives the same story told as "I was a hero" versus "I just escaped by luck"?

Well, when you come and tell the story, how do people react to you? And in particular how do your leaders react to it? Is it "Hey, great heroism over there, that was fantastic."

Sidney: You did what? Come here, I want to—

Steve: "Oh man, tell us more about how you almost went over the cliff!" You know, the mixed metaphors. "Tell us how you almost did so the rest of us could figure out better what to do in such circumstances."

Sidney: I think Allspaw has said it beautifully, that an incident is an unplanned investment, and if you don't see it that way as a leader, you are not getting a return on the investment that was already made on your behalf.

Gene: If I can just pause for a moment. So you see very collegial interaction between three very distinguished scholars—

Sidney: That's old veneer, that's old veneer.

Gene: Three and a half hours ago, it was like Worldwide Wrestling Federation, like there were chairs in the air—

Richard: Oh, come on, that's not true.

Gene: Yelling—

Richard: There were no bruises.

Gene: It was a very—when I say it was sort of a white-knuckled time, it was stressful, it was—

Sidney: Oh, that was only your problem, man. We were having fun.

Gene: John Willis and I, we were—

Steve: It was actually three on one.

Sidney: He's easily fooled, though.

Gene: What other advice, any other advice that you shared with me that you want to share with everybody else, in terms of unsolicited advice that you would give to us as a community?

Sidney: Don't be so easily fooled.

Gene: Don't be so easily fooled, yeah, thanks, man.

Richard: I recognize that there's a lot of expertise in this audience, and I think you have a kind of moral responsibility to share that expertise with the people around you, to help them become better prepared to deal with the

kinds of problems that they will deal with, even though you aren't going to be there at their side.

I think that responsibility needs to become part of what DevOps is. I think DevOps needs to go beyond tool chains and beyond repositories and become a kind of practice that involves people. What John has called "above the line."

To do this may require that we identify ourselves as people who practice DevOps rather than people who work for company x. That is, the practice of being a DevOps person has to be, actually, a kind of profession, a kind of skill, an expertise that exists apart from the particular employment that you're engaged in right now.

You've chosen to go to work where, essentially, the rubber meets the road, at the sharp end of the stick, at the cutting edge of things. And because you are working there you're going to have to encounter a lot of stressful situations, demanding circumstances, places where there's potential for real loss and the kinds of things that will keep you up all night. And I think you need to try and help the people around you who are learning about this understand what they are getting into and how to cope with that.

And I don't know exactly how to accomplish this except to say that the answer for this lies here in this room, not someplace else. It is not someone else's job to do this, it is not some other agency's job to do this, it is not your company's job to do this, it is your job to do this.

Sidney: Hear, hear.

Richard: And I think unless you find a way to band together, to find those common threads and the kinds of responsibility and moral activities that you engage in, that you run the risk of becoming regarded as just another group of technicians.

Sidney: Right.

Richard: And I think that would be a very great loss.

Steve: You mean like anesthesiologists?

Gene: Oh, I got to hear scholar jokes, like how do PhDs and doctors make fun of each other? I got to hear that.

Sidney: How do they?

Gene: It was funny in the moment.

Sidney: We love you, we do, we do.

Gene: Actually, why don't we end on that joke?

Sidney: Yeah, your foot is in your mouth already, so it's—

Richard: We're very grateful to Gene for inviting the three of us here. He's taking some risk in doing that, as you might have recognized, of all people who produce reliable performance under all circumstances, and Gene has taken some risk in bringing us together and building the kind of powder keg that exists with having three of us together.

But he's reaching for something, and I hope that you can identify what it is that he's trying to reach for, because I think that you'll have to embrace that and reach for it as well. I think that, more than any other technical aspect of these kinds of conferences, is what this is all about. And I encourage you to think about that and to pursue that over the year to come.

Gene: And I will have the closing words, and then I'll have an ask for all of you.

I had made the claim to this distinguished group on this panel that this is such an interesting and powerful community because you have all self-selected to be here. The fact that you're here signals that you see something bigger that we're aspiring to, often at great personal risk, and that you're building that coalition to overcome very powerful incumbent systems, so I guess here would be my ask to you.

If you are willing to be able to share some of these stories of uncertainty and making decisions when you don't know all the answers, I'd love to hear from you, just email me, Jess, and Perry, and this will be an obvious action item to create a section for that in DevOps Enterprise 2018. So with that, a round of applause for Dr. Richard Cook, the doctor, Steve Spear, and Sidney Dekker made sure that everyone knew that he doesn't have a PhD, he's only a medical doctor. Dr. Steven Spear, who was voted DevOps [inaudible] 2015; and everybody, he made sure that all of us knew that he was from MIT, not Harvard, not UPenn, and then Sidney Dekker, who reminded us all the time that he is a pilot, and has two PhDs, so thank you so much!

• • •

John: Yeah, I don't know about you Gene, but that was amazing.

Gene: Yeah, that was amazing. I've listened to it five times.

John: Yeah, so if I forced you to come out with one of your favorite takeaways of what was said on that panel, what would it be?

Gene: Oh, my goodness. Certainly, just the clarity of what he talks about the Andon Cord, just in terms of...the Andon Cord is a pairing of standardized work and the need that we always need to learn better methods, I think that was amazing. But I think the one that I think about a lot is the need to talk about the near misses. When he talks about "We can tell about the heroic

story," which reinforces one set of behaviors, and the story of "We were this close to grounding our ship" and how that actually reinforces a culture of "Tell us what you did that was within your control that helped reduce the risk," because those are relevant lessons for anyone in the same situation. I love the way they said, "Because we were white knuckling it, and trust me that there were things that were beyond our control, like the weather, the tides, etc.," I think that I viscerally understood something better through that story.

John, how about you? I'm going to challenge you with the same thing: what's the top moment for you?

John: It's ironic that this multi-year quest, "Gene, I want to do this, let's get this together," and I wanted to prove my hypothesis, but it was really something Dr. Richard Cook said at the end that was less about prove my hypothesis, that just stuck with me the rest of the night. In fact, even in the evening at beers, I was having arguments with people, trying to express this idea. And one of the things he talked about is an ethos, that DevOps needs an ethos, and he said something like, "Like a pilot of an aircraft or surgeon in an operating room, we expect whatever the long-term value of the expertise might be, we expect you to act in the moment as the most judicious, thoughtful, and prudent way." And so, he was getting to the point, "Do we have that ethos?" And then he said that he made a definition of DevOps, at the end, he said, "This is what I think DevOps is. DevOps is also a practice of building a community of people who do DevOps, and our goal is to teach DevOps to other people."

Combining the idea that, if we could think more about what is our ethos, as a profession, and also what is our role in that profession, just to go a little further, if you think about some of the conversation that I had that night, I just couldn't get it out of my head. Paula Thrash, J. Reed, all of them. I kept

wanting to hear feedback, and some people say, "Well, police officers have a well-defined fireman ethos. There's kind of ceremony, there's certain skills, but there's kind of a code," but there's also this idea that you're constantly training. There's a senior police officer and a junior police officer, and there's a role of which there's a nurturing. And I just thought that idea that we need to make sure we understand that a big part of our role is the nurturing of this knowledge that we keep aggregating.

Gene: And to reinforce that point, I think it was Sidney Dekker who said, "When you're a pilot, above all you are a pilot, and when you're talking to other pilots, you're talking to a fellow pilot, regardless of what/who is your employer, you are first and foremost a fellow pilot." To your point, I think what Dr. Cook is suggesting is that we are all fellow DevOps, fellow travelers, regardless of what company we happen to be employed by at that time.

It was an amazing day to spend with those authorities, and I think we had joked beforehand this could be a really historic moment to really get these figures together. I certainly felt like I was in the presence of something historic.

Module 8

Case Studies

John Willis: So Gene, let's start off with the first story is Courtney Kissler, who at the time was the VP of Technology at Nordstrom. It's a mainframe transformation story. And you know me. I'm an old mainframe person. But the story goes that every year leadership would discuss this old mainframe COBOL application. It'd been around for probably forty years or so, and it had long lead times, and it was just this anchor that everybody knew. It seemed, from the story as it went, that they would kind of get to a point where, "We'll table it." It seemed like it would happen every year. They'd have these discussions.

One year, Courtney decides to look at it and apply value stream mapping, a Lean technique, to look at it a little bit deeper. And as they go through the value stream map, they find out that really this mainframe COBOL application...the thing that was causing the long lead times...was there was a manual paper handoff that actually had to be transferred between floors. It had no validation. It just turned out that many times it was never even filled out. Sometimes it was filled out incorrectly. During the value stream map

where they uncovered this handoff, they were able to then actually fix it with a quick script, and it turns out all their assumptions about what was wrong were wrong, and it was really just this one little handoff.

Gene Kim: And you had a really interesting take on this that really I found very startling. That this is actually, in many ways, a Ladder of Inference story from Chris Argyris that we talked about last time.

John: When we talked about Ladder of Inference it was this idea that you observe stuff you selected, you put meaning through it, you make assumptions, conclusions, beliefs. So, you get the feeling that their leadership organization had got into this virtual cycle this feedback loop of just being convinced that anything that was wrong with the main...anything that happened on the mainframe had to do with COBOL and the mainframe.

And it was Courtney who basically...One of the things Argyris would say is try to force yourself out of that reinforcing loop. And the fact that she tried something alternative, like a value stream mapping against a mainframe. Why would you apply that to a mainframe? And then was able to uncover and break the belief cycle.

Gene: In fact, even as Courtney tells the story, I think she used these exact words: Everybody knew that these problems were caused by that, maybe she didn't say terrible, but that COBOL mainstream map. That thing that we've been trying to get rid of for a decade, but was just...We were just always been too busy.

In that phrase, she verbalizes this very strong belief that they have. That it was...It had to be the problem of the mainframe team. And I think the other thing that she exposes in that story was that, for the first time, they had assembled this group. Anyone even responsible for any element of delivery

of that service was assembled together. Probably bringing in a group of people that probably had never actually worked together before. And through this exercise they did reveal this kind of stunning problem that had nothing to do with the application but the field...the information they were asking for.

John: Yeah, I think we joked that imagine that there were the mainframe people..."Oh, wow. We're invited." And it's sort of...I wish she would have asked.

Gene: Yeah. I know.

John: I think the other thing, too, is we've looked at Courtney's journey there at Nordstrom. It seemed, too, they had evolved, and you have a shared gold star that I really like.

Gene: So, as a result of this workshop that I did for the COBOL mainframe, and this was for the cosmetic business office. That was just one of many places where they started using these Lean techniques. This birth of DevOps at Nordstrom was the in-store cafeteria systems, there was their e-commerce properties. I think they gained such confidence that reducing lead times for these services created business results.

And one of the outcomes was that in 2016 Courtney Kessler talked about how they actually integrated something into their bonus structures. In other words, in order to hit your bonus, you have to be able to reduce customer-facing properties lead time by twenty percent.

In *The DevOps Handbook* we actually got that a little bit wrong. We said that it was for all technology executives. But the reality is actually even more startling. It was not just for technology executives, but it was for the business executives as well.

And I loved that, because in my mind it's recognition that it's not a technology problem; it's a business problem. And so that incentive structure was applied to all company executives, regardless of whether it was business or technology.

So, I think that's a great example of systems thinking and shared goals. And I think all in this series, we put that under learning organizations.

John: Yeah, I think it's a great example of, like you said, learning organizations. If we even look at SAN guys, there's the mental model. The leadership had this view that really didn't match what was really...Their view was just the COBOL, and there had to be something wrong with the old code, and it was really something they couldn't see. And then systems thinking...

Gene: A problem they couldn't see. Straight from Mike Rother, the seminal book we talked about.

The other thing that occurred to me as you were telling the story was that COBOL workshop, the value stream map and workshop, that was actually done through an internal consulting capability that they created that would then go out to the various service and product teams. So in my mind, that's a great example of the Steven Spear capability three. There has to be some mechanism to translate local discoveries to create global greatness. I think that's another great example of that.

Another one of our favorite stories, without a doubt, is Target. This is the amazing journey that is commonly attributed to Heather Mickman, who was Senior Director of Development, and Ross Clanton, who was Director of Operations. It turns out that both Heather and Ross, while they were at Target, had rotated between many roles. They were Dev, Ops, InfoSec, and even architecture. The story I love is about Heather's area of focus for many

years, which was the API enablement project that she talked about in 2015 and 2016.

The business problem that she set out to solve was that any time that a team of developers wanted to access a system of record that they had never interacted with before, they would often have to wait six to nine months to be able to add, remove, change data. And that was because that was how long it would take for the integration teams to finally get around to them.

And so, what they did was create this next generation system of record where they basically took responsibility for copying all the data from the hundreds of systems of record, which included mainframes, John, and they put into this exotic new sort of technologies. But they had virgin APIs, so that any Dev team could on-demand add, change, remove data to sales data, inventory, store information, create orders, transfers—on-demand without having to wait.

What I loved about that is that they went on record saying that that capability enabled fifty-three different business initiatives to get shipped, including shipped to store, which if you're a retailer is a very strategic capability if you're competing against Amazon, the Pinterest integration, the Starbucks integration. All the in-store apps that support in-store employees.

I just love how her team grew in size year over year just demonstrating how strategic of a capability that was, because it helped increase...develop a productivity for the thousands of engineers at Target.

John: You know, a couple of things that I really love about the Target story in general. One is, early on, when we were doing a lot of DevOps discussions about the enterprise, and we really hadn't seen great stories in the enterprise. I remember it was a DevOps Days in Minneapolis where Heather and Ross did a "I'm Dev, and I'm Ops," and I was watching it from home screaming. I'm standing out of my chair "Yes, this is..." We knew that this was true, and

then to actually see a large, what we would consider a legacy organization, actually say, "We're doing DevOps."

Gene: They were one of the first presenters at DevOps Enterprise 2014.

John: Exactly, they were at our first DevOps Enterprise Summit Conference.

The second part of that story, too, is the story where Heather decided that she had to enable some really bleeding-edge technologies. So, we talked about DevOps and tools, and we talked about cultures and behaviors and all these things. They really mixed together very tightly. So, to move fast and to change the way she had to do and think about the API structure, she also knew that she had to rely on some tools. Like Netflix, we were using a very popular distribute database, a data storage system called Cassandra. Another queue management or pipeline tool called Kafka, a very popular web scale.

And the story, if I get this right, she went to the risk board, the people...the governance, and said, "I have to use these tools," and they said no. And she just knew that in order to actually create this transformation that was needed, [so] she did it anyway.

So it's one of my...I love a great "do now, ask forgiveness later" story.

Gene: And it turns out that that risk board was called the LARB meeting, the Lead Architectural Review Board.

So I got a chance to follow Heather Mickman around for a couple of days, and I remember seeing on her desk this certificate. It looks like something that was printed in Print Shop Pro on an inkjet printer, and it said, "Lifetime achievement award to Heather O'Sullivan Mickman for abolishing TEP and LARB." And so, of course, I had to ask, "What is TEP and LARB?" TEP is the technology evaluation process and LARB's the lead architecture review board.

So whenever you want to do something radical like use Redis or Cassandra or TomKat, something that the shared services organization had never done before, you would have to fill out the TEP form, and then that eventually gives you the right to pitch the LARB meeting. So, as described, you walk into this room, you have all the Dev architects on one side, the Ops architects on the other side, and they pepper you with questions, they start arguing with each other.

They give you thirty more questions to answer, and they say come back next month. Apparently, her reaction was, why do we even have the TEP and LARB? And she said, "No one could really remember." There was some vague memory of something unspeakably bad that happened fifteen, twenty years ago, but the exact details of what that disaster was had been lost in the mists of time. And her reaction was, "No one of my team should have to go through this. In fact, none of the thousands of engineers at Target should have to go through this."

And due to her relentless lobbying, they disbanded the TEP and LARB, earning her the certificate that hung on her desk when I was visiting her in 2015. So, I just love that because it really exhibits so much of that transformational leadership quality. She knew its critical capability that she had to develop at Target so Target could win in the marketplace. She was willing to take risks to do it because some understood the goal so well that she said, "If necessary, I'll do this anyway and ask for forgiveness later." I think that's such a great example of a transformational leader.

John: You know, the other thing that we love about the Target story is if you're looking and you consider yourself a student of DevOps, just following their journey because another great story is the Ross Clanton story of inventing this concept of a Dojo, a place that...He first presented that to us, a place where people can come and do immersive learning and accelerate

learning through the technology stacks, and the process, all the learnings, all the tools that we've talked about.

I think when we first met Ross he was showing us a presentation in probably an 800 square foot room, and I think it was last year. Both of us were able to visit at their campus outside of Minneapolis, I think it was Brooklyn, Minnesota. Fifteen thousand square feet. That journey to watch what they've done and how they've built it, and now it's just part of that Dojo the DevOps Dojo, has become part of the DNA...To go from just, hey, we're Dev and Ops showing up in Minneapolis to now it's a 15,000 square foot place where people are getting in queues, lining up to try to come into the DevOps Dojo.

Gene: In fact, I had a related story about just the gratitude that this Dev manager felt just being able to spend thirty days with Ross Clanton and the Dojo team. And just in my mind there's a couple of memes or practices that have entered mainstream. ChatOps, as pioneered by GitHub, but another one is the Dojo. That DevOps Enterprise, how many presentations were there about, here's our Dojo program, here's all the amazing things that come out of it. And that we can trace directly to his Target team.

I totally agree and that's another Steven Spear capability three. And I guess what I love about that is, I think what they figured out was it's not enough to give them books or send them to training. This is an experiential learning where we're going to solve a problem together. We're going to arm you with the best coaches and Ops engineering people to help you achieve your goals, and we're going to do it together. Oh, it's obviously a pattern that has resonated with the industry because it's so practical.

As we're discussing Target, I was thinking that...Another thing that occurred to me is there's a Theory of Constraints story here, too. What would cause Heather Mickman to believe with such certainty that this API

enabling project was so critical for Target. In one of the modules, we talked about how architecture can be the constraint. And when you look at thousands of engineers potentially waiting for integration teams to wire their services together, that is an incredible constraint.

In fact, one interpretation you could say is that fifty-three business initiatives were made possible by reducing architecture as a constraint. That was absolutely right on. So, I love the story just because they're drawing from, whether explicitly or implicitly or just guessing or just from experience, they concluded that architecture is where they needed to focus.

John: Yeah, I think there's another great story of bottom up that turned into a top-down story where they accelerated their learning by creating an executive view. You have a good story about that.

Gene: Yeah. Holy cow, yes. Now, I remember being invited by Heather Mickman and Ross Clanton to spend a day with their manager team. So it was four hundred of their top managers, and boy, was I surprised to see a whole bunch of familiar faces there. Courtney Kissler, who was at Nordstrom. Jason Cox from Disney was there. And a gentleman from Mark & Spencer was there. It was just amazing to see what they had planned. And so, only then did I learn that they were so moved by the DevOps Enterprise experience that they said, "We need to recreate that experience for Target executives and for our top managers."

Essentially, they had recreated the DevOps Enterprise conference but for Target leadership. I think what I started to appreciate in that moment was how ingenious these leaders are and how good at networking they are, the fact that they sought out all these people from fellow speakers and use that to help drive their agenda internally. I remember that there was a Target executive who said, "Heather, Ross, this was amazing. And I can say with some level of certainty that we've never had an executive from a competitor

come and talk about their experiences." And [I] think it just shows the level of savviness and the level of sharing that occurs within this community of people in the DevOps community.

John: Yeah, and the balance of risk too. Their ability to take these incredible risks, like building a Dojo, like using new technology and transforming the API structure...I love that story more than anything...is that a great transformation story, go invite other executives, other people that are doing what they do to present to their...

Gene: Even if it's your competitors. And just to punctuate something that you said in a previous module, the DevOps community is really a community of sharing, and this is such a great example of that.

John: You know, Gene, when you talk about sharing, I think back to this acronym Damon Edwards and I had created very early in the DevOps movement called CAMS: culture, automation, measurement, and sharing. What had happened was, there was the original DevOps Days in Ghent in 2009. It was somewhere less than a year later we ran one in Mountain View. That's actually where we first met. And the DevOps Days in Ghent had, like, forty-eight people. We had three hundred people at the first one in Mountain View in 2010. And after that event, Damon and I had been doing a podcast for years, a DevOps Café podcast, and we usually have guests, which you've been a guest sometimes.

But this time we just wanted to synthesize what happened. We literally had this "Oh, my God. How do we even explain to ourselves what that event, that three hundred-person event was like?" And we accidentally came up with this acronym: culture, automation, measurement, and sharing.

And you get the culture. We know now that it's a lot about behavior that creates culture. Automation clearly a big part of what we do. Measurement,

management of the telemetry, but the sharing, it was clear that even in 2010 that everybody was there...Nobody was showing up "I can't tell you this part about my business. I can't tell you that part." And in fact, within a year later, our good friend Jez Humble added CALMS, he added Lean. So now people refer to it as CALMS: culture, automation, or Lean, measurement, and sharing.

But anyway, the thing I think we both love about Target is it embraces all of those principles of DevOps—the culture, clear automation, and technology, leading edge technology. We didn't talk much about how you do telemetry and monitoring, but we both have been to their shop and it's extraordinary. But sharing, to take those kinds of risks to bring in teams like Jason and those guys to present, even their competitors to their executives, that's really what the DevOps community is all about.

Gene: John, I feel like sometimes we are the Siskel and Ebert of DevOp stories. I think we are in total agreement that Target has a special and unique place in the history of DevOps for so many reasons. John, I give Target a thumbs up.

John: It's a two thumbs up for me. Gene, you know the story that we both love is Scott Prugh. He's the chief architect and VP of development and product operations at CSG. People that don't know what CSG...Basically, they are one of the largest printing, billing companies...They print all of your bills. In fact, the way Scott would describe it is if you get a paper-based bill from your cable company, they're the ones that did it. The thing about Scott and CSG, and some of the other people that work with him, they've been at every DevOps Enterprise Summit from 2014 all the way through, and you get to see...You know, just like we've talked about Target and we love their journey, you can see this journey year over year. And some people come up

to conferences, and they'll give a presentation, and they'll say here's where we are and we're so great, and then you never see them again.

Gene: Ha!

John: With CSG, you get to see what they look like, and then a year later you actually get to say, "Did they get better?" And they seem to get better every year. But you recently told me a story that they haven't actually presented anywhere in...And I love this story.

Gene: Yeah, and I think this is just one of the many reasons why I love the CSG story and what Scott Prugh and his team have done. I think it's a story about engineering greatness. So, what he has talked about is just how difficult of an architecture of where they're focusing their transformation on; at its core, it's a mainframe application, but it also involves twenty other technology stacks, .NET thick line, .NET thin line, j2ee—you name a technology, it's probably in there somewhere. And it's funny how mainframes keep coming up here, but at its core, it was an application that was created forty-five years ago, and then it was spun out.

One of the challenges in the focus areas for the CSG team is so many of the features have to go through the mainframe team, and the mainframe team is the only team that can actually make the needed changes to functionality. A reality is that the mainframe workforce continues to shrink. So what has been talked a lot about is how they've been able to reduce code deployment lead time from fourteen days down to less than a day, how they have been able to increase reliability. But what hasn't been talked about so much is what they're doing to ameliorate some of these problems that they face around the mainframe.

Talk about some of the high-level goals of these internal engineering efforts. One is start migrating from the VSAM database to Db2 in

progressive stages, and so I didn't know this, but the VSAM database...This goes way back where it's almost a key value store. You have to manually index by column numbers, and when you're doing this in Assembler, there's a lot of manual effort just to get to the right column. You're smiling.

John: Yeah, I'm smiling because that's...In the first part of my life, that was something I had to deal with, those old record structures, and they had variable lengths. It just gives me the willies right now.

Gene: So one of the problems was that every time they wanted a new report the mainframe team would have to write a new feature. So, they wanted to bring continued integration and test-driven development practices to the mainframe platform to increase a safety of the coding environment. And they also wanted to start converting over 4 million lines of assembly code to Java one module at a time. One of the end outcomes would be that any Dev team would be able to directly query Db2 instead of having to go through the application. So the Heather Mickman API [inaudible] project, how do you decouple the Development team from the mainframe teams?

So, one of the stories that had just a lot of personal meaning to me was they paired a Java programmer, one of the lead programmers, with one of the mainframe Assembler programmers. They wanted to see what they could do in terms of experiment of doing the conversion. The thing that blew me away was after three months, they had turned 130,000 lines of assembler code manually indexing in to the VSAM database into 800 lines of Java. It's just amazing. So, this is Java code that I've seen. It's like, oh, hey any developer could probably understand this and maybe even modify it. So I got an opportunity to meet that team, and I just wanted to understand and learn, What did it feel like for that mainframe programmer? You know, was it fear? Was it excitement? Was it dread? Fear for his job? Right? I mean, I didn't know.

Here's some quotes that I'll never forget. He said, "You know, trust me, even though I volunteered for this project, I brought my share of baggage because I knew that there's no way that anyone could write something as fast as the Assembly code and that modernizing wasn't going to work." He said, "This is our fifth attempt. A lot of money spent on attempts to replace never worked; this one probably won't either." He said, "There was a period of intense frustration and bewilderment, but it was amazing to take the best parts of open systems and .NET and make it work on the mainframe." He said, "It was exhilarating watching thousands of lines of code disappear. That night I couldn't sleep and emailed my manager saying this is going to work. As a CSG shareholder and someone soon to retire, I know that doing this is going to create long-term value for the organization."

I've just thought this was...I mean, I still get chills telling this story about how, I mean, how many ways does this give me chills, one is that even the pairing of the team. It's a young Java engineer paired with an engineer who's probably twenty years older, and he's sharing his experience learning, and there's a personal gratification he got out of it, and his absolute confidence knowing that this was creating genuine value for CSG.

John: You know, and to me when you first told me this story...It didn't...It wasn't the thing that it was a mainframe story that got me excited; it was the story that I've been watching Scott Prugh from 2014. When he talked about how they were doing test-driven development, automated test-driven development, and then in '15 they come back, and they talk about this pipeline that's just...And after about three or four years, they've invested so much in...You know, Scott even talked in one of his earlier presentations about how he went to the supply, the floor, to meet with the actual manufacturing people to make sure he was going to do Lean principles, but he wanted to hear their thoughts.

So, you watch this journey and understand that they've built this resilient pipeline with a lot of investment and DevOps, or we would call this the first way, if you will, from *The DevOps Handbook*. The story, to me, was that without all that, the idea of going to leadership and saying, "Hey, I want to go ahead and convert a mainframe application to Java," people get fired for that suggestion because they never work and it sounded like they never work before. But the story to me was there was so much trust in the system that because they, you know you had to do...Everything had to be done...You know, you had to build test-driven development, you had to have your backend tests—all those things. And so it seemed to me—and I've talked to Scott about this actually this year—it was this notion that Java, Assembler mainframe, Python, and Ruby—it kind of didn't matter because they had trust in the resilience of the system that they built.

Gene: In fact, it's interesting you say that because Scott, he's very clear about that. He said his...There's a category of thought called bi-modal IT that says you change your front-end systems, and you apply DevOps there, and you don't change your backend systems that are the mainframe systems. And he comes up with a different category. He said, "You have a category that you can depend on, that you can patch, you can change, you can configure it, you can secure it, and the other categories are those that you have to take into the parking lot and smash with a sledge hammer." Which he actually had on video showing how they actually, finally able to do that for a class of servers. I can hear him saying that. It's like, I don't care what language it's in, if customers depend on it for speed or quality, you know we have to apply modern engineering practices.

John: Yeah, and I think that again, if you look at what we've covered throughout, the principles of Lean and some of the Theory of Constraints

and safety and learning organization, these are the principles that you apply beyond the technology.

Gene: Right, it doesn't matter what technology it's on.

John: So, the importance of what we've been trying to talk about, all throughout this work here, is why this is so important. Because when you get that stuff right in your organization, then things like mainframe conversions to Java become kind of non-incidents.

Gene: Right, it can be done safely. In fact, I just want to...I got a list of things from Scott's perspective. Here's the value of doing it, that I think that every business and technology leader would appreciate. He said the value is that now this type of work can be preformed by a much broader population of engineers. It's not the shrinking group of mainframe teams that only they can do it. The changes can be made more quickly and safely, because they have the safety net of automated testing behind it. Business value can be quickly delivered independent of the underlying platform. And, by the way, as a result of doing this, it's still running on the mainframe, but it's just running in Java, which reduces operating costs. And so, I love the fact, this is what world-class engineering organizations do.

Gene: Yeah, and I think anybody who would be listening to this right now that have mainframe applications, it's not that they don't want to convert them. Right? They all want to convert them. It's the fear of converting them. And again, I think the messaging here is that if you invest in these kinds of processes and ideas, built on Lean and safety, it enables you to do this in a way that is safe.

John: So, another great story, Gene, is...It's got to be Jeffrey Snover. Jeffrey Snover is Technical Fellow over at Microsoft.

Gene: One of eleven individual contributor technical fellows.

John: Yeah, it's amazing. And it's more amazing when you get to hear his story. He presented it at DevOps Enterprise Summit 2015 in San Francisco. A presentation called, "The Cultural Battle to Remove Windows from the Windows Server."[1] But it was a much bigger story than that. It was about somebody who was a leader, who basically fights and transforms in a large organization, works really hard. I actually got to meet Jeffrey many years ago where we did a DevOps Café podcast with him.

What was interesting, and Damon Edwards, my co-host, was always like, "We need to get this guy on," and I'm like, "A Microsoft guy?"

And one day he hands me his bio, and it's...he was at Tivoli. I did a ton of Tivoli work. And so before we even did the podcast, we started comparing notes. He actually worked on a large project, we must have just missed the whole time that I worked heavily with. It was called Tivoli for Windows. So we geeked about that.

And Microsoft did an industry hire to bring him there. He got an executive role and immediately saw, coming from a Tivoli background, a very command line cis-admin motivate...So Tivoli was acquired by IMB. But a very cis-admin command line motivated framework, he goes over to Microsoft. So naturally he starts thinking about, "Well, I think we've got it wrong." And he actually starts working on a paper, this kind of monad manifesto, which ultimately becomes PowerShell, But, at one point, as he told the story, I think in the presentation but certainly on the podcast, is that at first they told him, he had to take, at his level that they hired him, "You can't do this." And he was like, "What level do I have to be?" And he took a demotion to do that, continued the work on it. It's almost, that guy is crazy.

He tells the story where I think, one of the versions I heard, he used some curse words. He said, that at one point, Steve Balmer walked in his office and

said, “What the explicit about Windows?”

Gene: Yeah, what, what don’t you understand about Windows, Jeffrey? [*laugh*]

John: Yeah, but he kept moving on, and in his 2015 DevOps Enterprise Summit presentation, he even implied that people would be like, “Oh, he’s over by the coffee pot, wait until he leaves.” Because it was career ending to be associated with him.

Gene: Or even be seen near him.

John: To turn out to see him become a Fellow and then the coming out was all that work of transformation and just putting your head down and knowing that this was right for this organization, against a lot of bad odds, and to tell a story of he is instrumental in taking the Windows out of Windows server, which was basically something called the Nano Server, which was actually a server where you didn’t have all the GUI Windows. Which was critical to get into cloud and all the other new technologies.

Gene: In fact, I mean, just to geek out for a second I think one of the things that he shared so proudly was that the boot time for Windows Nano Server was actually faster than Linux. So, an incredible engineering achievement. I love his story for so many reasons. One is, like you, I have a tremendous amount of admiration for his contribution to the industry, and he was lured to Windows because he could see what Windows did for the desktop world they could do potentially for the server world as well. And what I so appreciate about the story was that he had this story of almost redemption.

He took the demotion to do what he knew was right, and then as he told the story, you see it against the backdrop of some of the most recognized properties that anyone in technology could see.

So, as he is navigating this chasm and this maze that is the Microsoft organization at the time, and the first place he gets a wind is Exchange Server Group in 2007. So, that's the first place a team sees the value in what he wants to do in terms of a command line way to manage mission critical infrastructure. And then the .NET group becomes PowerShell starts showing up through there. SQL Server, the Windows Server 2012. And what I love about that story is that here is a person who is so clearly technically expert but also is someone who is savvy enough and relentless enough to go to, as he tells it...it seems like, every major product group, and is continually pitching and finds those early winds and ladders up in terms of his support base, to now where everyone recognizes that he is, without a doubt, one of the top engineers in Microsoft that has sixty thousand engineers in it.

John: Well, and you know, one of his early wins along that journey was...in fact, one of the places I first started working was when I was at Chef. And it was one thing to tell Marcus off to go to command line, but there was another thing when they saw this whole emergent where a lot of their customers were moving to infrastructure as code, this idea that you could build code to describe how you want to install software.

Gene: Right.

John: And Microsoft was watching this happen, and I think that even today, people would say that Microsoft has probably sixty percent of most large data center, larger enterprises; forty percent is Linux. They were watching that forty percent got eaten by this model. And Jeffrey actually started working very closely with the Chef people and actually created the first really true configuration management. This was after he had done PowerShell. It was a desired state configuration management.

Gene: DSC.

John: And that ingrained him into the opportunity to work with a lot of those groups.

Gene: You know, there are two things that I think also are noteworthy. I say that with significance. One is, he has been also such a vocal proponent of, as he calls it, "surviving the transformation" or "surviving the transition." And how many times in DevOps Enterprise have people like Scott Nasello from Columbia Sportswear, he said, "We had a large part of our work group come from Windows, they came from help desk, and we had to show them that the days of the click-next admin were coming to an end. They actually brought in Jeffrey Snover, and he has been tireless in describing his own experiences in terms of, he saw something around automation back in his Tivoli days, and he saw something in Windows. He became the very best expert at it to the point where he was recruited by Microsoft. He knew that he was sitting on knowledge that was important and this was the way the industry was moving, and in the same way, he is saying every Windows administrator, IT pro, who isn't picking up automation skills is at the risk of becoming irrelevant.

That human part of Jeffrey Snover, that is just amazing to watch.

John: You know, one of the things, just to get a little geeky here for a minute. Most people that have never worked with PowerShell that come from assistance backgrounds get very annoyed in the order of the verb, the command and the parameters. And I've talked to Jeffrey—it was by design. Because in a Windows world, I think the story is that there are these leaders, that it's no coincidence that we talk about them, that make these kind of decisions that just seem, in time, just the right decision. And I think, even that subtle—I mean I've watched people on Twitter that have been doing a Linux-based system forever and now all of a sudden they have been tasked to do some Microsoft and PowerShell, and they're just on Twitter cursing,

"Oh, I hate PowerShell! Oh, my God!" And then at some point they are like, "You know what? I really like this."

Gene: And PowerShell now is not just Windows. It's on Linux, it's running the cloud, it's the interface to Azure. You know, the other thing that I say, with some embarrassment, is the Monad Manifesto. As I have been learning the Clojure programming language, it's a functional programming language, it's meant to be immutable. Even the word Monad that he chose has a lot of significance. It arises the notion of, how do you contain the side effects of programs into things called monads? So, once again, Jeffrey was way ahead of the curve, and I'm still catching up to him, and I'm still trying to learn exactly what a monad is, and I've spent hours trying to understand... have flickers of understanding, but I know Jeff knows what a monad is.

Gene: John, one of the things that I want to underscore is that I think DevOps is so urgent and important, both from a technology perspective as well as from a business perspective. Chris Little, a good friend to both of us, he once told me...We were sitting outside DevOps Days London 2011, and he said, "Of course, in fact, we've just seen the tip of the iceberg in terms of the value that DevOps would create." And he went on to say that...Incredibly, he said, "We're just at the beginning of a twenty- to thirty-year golden age." He said that with such certainty and confidence that I really want to know why he believed that.

And that's when he introduced me to a book called *Technology Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages*.[2] So this was written by Dr. Carlota Perez, and I think that this has gotten a lot of interest from many circles because what she did as part of her doctoral work was explain and posit some causal reasons why bubbles and busts happen. And so many people have noted [Dr.] Perez wasn't the first, but Dr. Perez, she noted that there have been four macroeconomic cycles

that have happened in the last couple of centuries. The first was the industrial revolution, the second was the age of steam and railways, the third was the age of steel and heavy engineering, the third was the age of oil, autos, and mass production, and the fourth is what she calls the ICT revolution. The information, communication, and technology revolutions.

And so these happen every fifty to seventy years, and so, I think, many people have observed that. But what Dr. Perez did was say that the reason these cycles happen is because of five things. First, there is some critical factor [in] production that becomes suddenly very, very cheap. So, whether that was ship production or manufacturing, it creates this surge of infrastructure being built. It also fuels a laissez-faire period of wrenching innovation followed by a bubble. So, this is when there is a boom as all sorts of inventions are created to exploit this new technology.

And then there is the fourth phase, which is a post-bubble recession followed by a reassertion of institutional authority. Could be additional government regulations and so forth to make sure that that bust doesn't happen again. And then Dr. Perez says a period of consolidation and widespread gains and the improved productivity from the new technology.

So, what is really remarkable about Dr. Perez's work is a split that she makes. Basically, everything before the bust is something she calls the installation period. And that's the period driven by financial capital. So, think about Wall Street or investor money. This is fueled by people out to make a buck, you could say. And then after the bust is the deployment period, and that is fueled by not financial capital but production capital.

So many stories we hear at DevOps Enterprise, whether its Capital One, or Nordstrom, or Target, these DevOps initiatives, they didn't go out to raise money from venture capitalists. They were doing it from operating capital. And what's remarkable is that Dr. Perez says it is this deployment period where we are drawing upon production capital—that is where the majority

of value will be created. That has been the case for the four previous economic cycles, and this is what is likely to happen with the incredible factor in technology that something that was so very expensive is now so much very cheap.

And so, I think that is why Chris Little says with such confidence that we are at the beginning of a golden age where DevOps is not used by the FAANGS—the Facebook, Amazons, Apples, Netflix, and Googles—but is now by every large, complex organization, the largest, and every industry vertical.

Incidentally, what I find very interesting is that the period of installation, the financial capital period, has often been marked in previous eras by an ever-increasing rich-poor gap. Where as the period of production capital is really where the tide lifts all boats, and this is where the rich-poor gap gets shrunk. And so, I think one of the things that we've seen happening in the last...previous decade is this exact phenomena. I think a natural side effect of this second period that we're entering is a more just society. So, I think that's something worth noting as well.

The last thing I want to point out here is that there is something very disruptive going on now. The whole notion of digital disruption. There's a graph from Goldman Sachs that was very striking to me. They basically show that Amazon is as disruptive as Walmart was. So, on the graph, it shows on the y axis the percentage of all retail transactions from the two organizations and on the x axis is the number of years since their IPO. And we can see on the chart that they're tracking almost exactly. In other words, Amazon is as disruptive as Walmart was. And so, history doesn't repeat itself exactly, but it certainly rhymes.

I think what we're seeing so much today, now more than ever, is that DevOps is helping enable real digital disruption, where organizations like Nike in 2017, they went on record saying that to deal with the rapidly

changing economic health of retailers they are actually going to sell their product in the Amazon Marketplace, and they projected $300 million to $500 million of net new revenue to come through the Amazon channel. And that is not just a brand effort, but that is primarily a software effort.

And so, I think, all of this in my mind shows just how important all this work is. That the stories of the unicorns like Google, Amazon, Facebook, that is where a lot of these techniques were forged, but ultimately we are entering an era where DevOps, digital [inaudible], the elevation of technical practices, that will affect every industry. And that's not just commercial industry; that's also every government agency, it's every not for profit. Just as Dr. Perez predicts, I think so much of the DevOps enterprise tools are so notable just because of the nature that they are funded by industrial capital, not by venture capital. I also do believe Chris Little now… I think there's every reason to believe that DevOps will be a very critical part in where the next surge of productivity will come from. Not just for years, but potentially even for decades.

John: You know, Gene, I love that paper, and the notion that the transformation is going to come from industrial capital. Earlier this year, I saw a presentation from Marty Chavez, he's the CFO of Goldman Sachs. And I like to repeat this—the CFO of Goldman Sachs. There's been a couple of articles that Goldman Sachs wants to become the Google of Wall Street, and there's actually a presentation where he goes back to his alma mater, Harvard, where he tells a really in-depth story of Goldman Sachs has incredible IP around financial risk, and from his perspective, everybody wants access to that knowledge. He said imagine when Google was starting that if you had to call Google—I'm paraphrasing—but if you call Google and said, "Hey, Google, I need to know...Yeah, the help desk—how many cats are there in Pennsylvania? Like 70,000,332." He said that's what we do at Goldman Sachs right now. So he talks about we need become...this

transformation. So to your point, they invest industrial capital into this idea of transforming how they make risk IP and knowledge available in a Google-like way.

And even to technically riff for a second, in this presentation, not only does he talk about that level of transformation, which is, of course, including all these properties of what we call DevOps and all the things that you have to do to improve to get to that state. But he goes on, and he talks about we're becoming a cloud first, we're becoming open source. He even starts naming some of those technologies like Kafka. Again, this is CFO, so this is when...Industrial capital is moving to a point where the CFO is describing a legacy structure of how they did financial risk and IP and share that with our clients to talking about open source and cloud. That is evidence.

Gene: I love that, and it reminds me of Richard Fairbanks, the CEO of Capital One. He said in order for us to win in the marketplace, we need a world-class technology organization, just like what the CFO of Goldman Sachs is saying. And I also appreciate the irony is that Goldman Sachs, they're using their internal industrial operating capital to achieve this transformation, and often for many of us, there's a gateway to financial capital. But they're not raising money to do this; they're doing this to help with the long-term interests of Goldman Sachs. So, I think that is another great example of this.

I think for all these reasons, I absolutely now believe our friend Chris Little when he says we are now at this beginning of the value that DevOps will create. And if we expand DevOps to dynamic learning organizations, I mean, in my mind, it is beyond question that we are on the cusp of an incredible surge of productivity that will last not necessarily for years, maybe even decades into the future.

So Dr. Carlota Perez's work is very persuasive. It's not the easiest book to read, but I found it very rewarding. And, to sum this all up, I think now more than ever, DevOps is important and the stakes of any organization doing this right can be huge.

Module 9

Conclusion

John Willis: So, Gene, what an amazing journey. I think from when we first met, one of the things that I've really enjoyed is we share stories. You know, we share stories at DevOps Enterprise Summit with people, we hear stories, we exchange the stories. You'll send me a link to a story or you'll call me or I'll call you and I'll tell you a story. Then, as we've talked from the beginning of this work, this idea of let's do this *Beyond The Phoenix Project*. I think it was a great opportunity for me and you to share a lot of those stories with everybody else.

Gene Kim: And, even through the sharing stories with each other, I think we were talking about some common stories we both knew, and there were so many places where I learned of an aspect of the story that I had actually never heard before, like specifically the Heather Mickman Target story. By the way, it was really exciting to hear where exactly the CAMS model came from—the culture, automation, management, and sharing—that it was a

direct side effect of that amazing DevOps Days Mountainview 2010. That is just—I never heard that before.

John: Yeah. I think along those lines, all those topics, like you said, there were some that I had not heard, and there were others that I had my own, if we use the word mental model again. I had this mental model of how I thought the story, and then I heard you either tell it or add your mental model of it, and like, "Oh, wow." That's...

Gene: That's a better story.

John: Yeah, its additive to what I, or I didn't really under that aspect of it. Absolutely.

Gene: For me, one of the high points has been this journey of stitching together all these various movements that DevOps draws upon and putting them together in a single timeline. I've got to be honest, when we first brainstormed about this project many years ago, I think I might have smiled and nodded. You said, "From Darwin to DevOps." And I smiled and nodded. And through the hundred-plus hours of work of the preparation and the recording of this, I've absolutely changed my mind. It went from somewhat of a certainty to absolute certainty.

I love the fact that you've been able to stitch a story that includes Dr. Goldratt, Dr. Deming, Lean, safety culture, and even learning organizations and show that there is a lineage that goes back and that they are mutually supportive movements that are now coming under this umbrella that we would assert is going to be called Dynamic Learning Organizations. That is something that I have learned so much just in the recording of these sessions.

John: Yeah, I think also it helped us codify, at least, our view of all this. I think we've made the point that a kind of Lean, safety, and learning organizations are three major influences on DevOps and transformation and then made the point that there was some "Shoulders of Giants" discussions, Dr. Goldratt, Dr. Deming, that clearly evidence of impacting all three.

Now, absolutely.

Gene: And by the way, I have to say that I loved how you have pointed out all the places that Deming shows up in history. He shows up in the foreword of Dr. Senge's book. He shows up in the acknowledgments of all these other works we are familiar with. I had read these books, but I had never seen them. I mean, I glossed over them, and it's just amazing to see some of the impact that Dr. Deming has had.

John: Yeah. You know, a very succinct takeaway for me is when we were working on *The DevOps Handbook* there was...You know, before we had completed it, you had recommended that I read Mike Rother's *Toyota Kata*, and then a little after that you recommended Dr. Spear's *High-Velocity Edge*. I looked at those two books as you should read both, and I'm a big fan of Rother's *Kata*, but I didn't really understand why you were so excited about Spear's work. As we've gone through this body of work, I've had better appreciation because his concepts show up everywhere.

You'll point out one of his capabilities here, and I'm like, okay. I have a better appreciation and certainly working with him on the four-hour panel at the DevOps Enterprise Summit, to get to hear him directly, the way he's codified learning. I have a better appreciation how that relates to all his work.

Gene: I love it, and I think one of things that I certainly appreciate more than ever now is how fortunate that the DevOps movement can benefit from

all these amazing bodies of work. Some of them that barely acknowledge each other. Like Lean and safety culture. And yet, we get to take advantage of all their learnings and their teachings. One of things that I'm just incredibly optimistic about is that the DevOps community, more than any other community that I can think of, is not inwardly focused. We are always actively learning and seeking other models and lessons so that we can help better achieve our own goals and aspirations.

John: Yeah I think the strongest message of this is we've tried to codify all the things that we think have been influences of DevOps, but it's really on you, the listener, to really take the challenge and follow up...many resources, these things...We have to keep, continually...Everybody has to roll up their sleeves and, like Dr. Cook said, "DevOps's role is to teach other people DevOps."

Well, what that really means is DevOps's role is to understand some of these things like Lean, safety, learning organization, and just keep adding more and more disciplines and new disciplines as they come up and shaking them out, seeing if they fit, and trying to apply them to what we're...this learning phenomenon that we call DevOps.

References

Module 1

1. Eliyahu M. Goldratt and Jeff Cox, *The Goal: A Process of Ongoing Improvement* (Great Barrington, MA: North River Press, 1992) Kindle location 504–506.
2. Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Upper Saddle River, NJ: Addison-Wesley, 2003).
3. John Willis and Damon Edwards, "Episode 62: Guests: Mary and Tom Poppendieck," August 17, 2015, on *DevOps Café*, podcast, MP3 audio, 64:01, https://itunes.apple.com/us/podcast/devops-cafe-ep-62-guests-mary-and-tom-poppendieck/id371931111?i=1000349854987&mt=2.
4. Gene Kim, Patrick Debois, John Willis, and Jez Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (Portland, OR: IT Revolution Press, 2016).
5. Ben Rockwood, *DevOps Demystified: An Introduction to the Ideas That Are Driving DevOps*, CuddleTech.com, http://cuddletech.com/slides/DevOps-Demystified.pdf.
6. Mike Rother and John Shook, *Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA* (Cambridge, MA: The Lean Enterprise Institute, 2003).

7. Jez Humble, Chris Read, and Dan North, *The Deployment Production Line*, https://continuousdelivery.com/wp-content/uploads/2011/04/deployment_production_line.pdf.
8. Steven J. Spear, *The High-Velocity Edge: How Market Leaders Leverage Operational Excellence to Beat the Competition* (New York: McGraw Hill, 2009) Kindle location 559.
9. Andrew Shafer, "The Learning Organization—MoDev," SlideShare.net, published on December 12, 2013, slide 12, https://www.slideshare.net/littleidea/the-learning-organization-modev.
10. Kurt Vonnegut, "Kurt Vonnegut on the Shapes of Stories," YouTube video, 4:36, posted by David Comberg, October 30, 2010, https://www.youtube.com/watch?v=oP3c1h8v2ZQ.
11. Eliyahu Goldratt, *Beyond the Goal: Eliyahu Goldratt Speaks on the Theory of Constraints*, read by the author (Gildan Media, 2006) Audible audio ed., 5 hrs, 46 min.

Module 2

1. Eliyahu M. Goldratt and Jeff Cox, *The Goal: A Process of Ongoing Improvement* (Great Barrington, MA: North River Press, 1992).
2. Goldratt and Cox, *The Goal*.
3. John Willis, "Operations is a Strategic Weapon," SlideShare.net, posted by John Willis, October 7, 2011, https://www.slideshare.net/botchagalupe/operations-is-a-strategic-weapon.
4. Ron Kohavi, Thomas Crook, and Roger Longbotham, "Online Experimentation at Microsoft," paper presented at the Fifteenth ACM SIGKDD International Conference on Knowledge Discovery and Data

Mining, Paris, France, 2009, http://www.exp-platform.com/documents/exp_dmcasestudies.pdf.

5. Eric Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* (New York: Crown, 2011).
6. Eliyahu Goldratt, *It's Not Luck* (London: Routledge, 1994).
7. John Allspaw and Paul Hammond, "Velocity 09: John Allspaw and Paul Hammond, "10+ Deploys per Day," YouTube video, 46:21, posted by O'Reilly, June 25, 2009, https://www.youtube.com/watch?v=LdOe18KhtT4.
8. Eliyahu Goldratt, *Standing on the Shoulders of Giants: Production Concepts versus Production Applications: The Hitachi Tool Engineering Example* (Goldratt Consulting, 2006).

Module 3

1. Mark Burgess, *A Tiny Overview of CFEngine: Convergent Maintenance Agent*, paper presented at the Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO 2005, http://markburgess.org/papers/tiny_intro.pdf.
2. Mark Burgess, "Computer Immunology," paper presented at the Proceedings of the 12th Systems Administration Conference (LISA 1998), https://www.usenix.org/legacy/publications/library/proceedings/lisa98/full_papers/burgess/burgess_html/burgess.html.
3. Frederick Winslow Taylor, *The Principles of Scientific Management* (New York: Harper & Brothers, 1911).
4. W. Edwards Deming, *Out of the Crisis* (Cambridge, MA: MIT Press, 2000).

5. Peter Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization* (New York: Doubleday, 1990).
6. W. Edwards Deming, *The New Economics for Industry, Government, Education* (Cambridge, MA: MIT Press, 1993).

Module 4

1. James P. Womack, Daniel T. Jones, and Daniel Roos, T*he Machine That Changed the World: The Story of Lean Production—Toyota's Secret Weapon in the Global Car Wars That is Now Revolutionizing World Industry* (New York: Free Press, 1990).
2. "NUMMI," *This American Life*, March 26, 2010.
3. *Time*, April 13, 1987, http://content.time.com/time/covers/0,16641,19870413,00.html.
4. Steven Spear and H. Kent Bowen. "Decoding the DNA of the Toyota Production System." *Harvard Business Review* 77, no. 5 (September–October 1999): 96–106.
5. James P. Womack and Daniel T. Jones, *Lean Thinking: Banish Waste and Create Wealth in Your Corporation* (New York: Free Press, 1996).
6. Jeffrey Liker, *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer* (New York: McGraw Hill, 2004).
7. Mike Rother, *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results* (New York: McGraw Hill, 2010).
8. Spear, *High-Velocity Edge*.
9. *If Japan Can...Why Can't We?* NBC, produced by Reuven Frank, aired 1980.
10. Mark Schwartz, *The Art of Business Value* (Portland, OR: IT Revolution Press, 2016).

11. Steve Blank, *The Four Steps to the Epiphany: Successful Strategies for Products That Win*, Second Edition (K&S Ranch, 2013).
12. Mary Poppendieck and Tom Poppendieck, Mary, *Lean Software Development: An Agile Toolkit* (Upper Saddle River, NJ: Pearson, 2003).
13. Lean Enterprise Institute, "Lean Thinking and Practice," Lean.org, https://www.lean.org/lexicon/lean-thinking-and-practice.
14. Valerie Reitman, "Toyota Motor Shows Its Mettle After Fire Destroys Parts Plant," The *Wall Street Journal*, May 8, 1997, https://www.wsj.com/articles/SB863043244663561500.
15. Eric Ries, *Startup Lessons Learned* blog, http://www.startuplessonslearned.com/.
16. Donald G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development* (Redondo Beach, CA: Celeritas Publishing, 2009).
17. David J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business* (Sequim, WA: Blue Hole Press, 2010).
18. Steven Spear. *The High-Velocity Edge: How Market Leaders Leverage Operational Excellence to Beat the Competition* (New York: McGraw-Hill Education, 2009) Kindle Locations 63–67.
19. Jim Collins, *Good to Great: Why Some Companies Make the Leap...and Others Don't* (New York: HarperCollins, 2001).

Module 5

1. Alan Cooper, *Inmates Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity* (Sams Publishing, 2004).
2. Kevin Behr, Gene Kim, and George Spafford, *The Visible Ops Handbook: Implementing ITIL in Four Practical and Auditable Steps* (IT Process Institute, 2004).

3. Charles Perrow, *Normal Accidents: Living with High-Risk Technologies* (Princeton, NJ: Princeton University Press, 1999).
4. Sidney Dekker, *Just Culture: Restoring Trust and Accountability in Your Organization* (Boca Raton, FL: CRC Press, 2017).
5. Richard Cook, "How Complex Systems Fail," Cognitive Technologies Laboratory, University of Chicago, 1998, http://web.mit.edu/2.75/resources/random/How%20Complex%20Systems%20Fail.pdf.
6. David D. Woods, Sidney Dekker, Richard Cook, Leila Johannesen, and Nadine Sarter, *Behind Human Error*, Second Edition (Boca Raton, FL: CRC Press, 2010).
7. John Allspaw, *Trade-Offs Under Pressure: Heuristics and Observations of Teams Resolving Internet Service Outages*, Master's Thesis, Lund University, Sweden, September 7, 2015, http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8084520&fileOId=8084521.
8. Micheal T. Nygard, *Release It!: Design and Deploy Production-Ready Software* (Pragmatic Bookshelf, 2007).
9. Sidney Dekker, *Drift into Failure: From Hunting Broken Components to Understanding Complex Systems* (Boca Raton, FL: CRC Press, 2011).
10. Jesse Robbins, Kripa Krishnan, John Allspaw, and Tom Limoncelli, "Resilience Enginerring: Learning to Embrace Failure," *acmqueue* 10, no. 9: September 13, 2012, https://queue.acm.org/detail.cfm?id=2371297.
11. Sidney Dekker, *Just Culture: Balancing Safety and Accountability* (Hampshire, England: Ashgate Publishing, 2007).
12. Sidney Dekker, *Field Guide to Understanding Human Error* (Hampshire, England: Ashgate Publishing, 2006).
13. Sidney Dekker, *The Safety Anarchist: Relying on Human Experience and Innovation, Reducing Bureaucracy and Compliance* (New York:

Routledge, 2018).

14. John Allspaw, "Blameless Postmortems and a Just Culture," *Code as Craft* blog, May 22, 2012, https://codeascraft.com/2012/05/22/blameless-postmortems/.

15. Bethany Macri, "Morgue: Helping Better Understand Events by Building a Post mortem Tool – Bethany Macri," YouTube video, posted by info@devopsdays.org, 33:34, October 18, 2013, https://vimeo.com/77206751.

Module 6

1. Edwards Deming, commentary on Peter Senge's *The Fifth Discipline* (1990).

2. David Garvin, Amy C. Edmonson, and Francesca Gino, "Is Yours a Learning Organization?" *Harvard Business Review*, March 2008.

3. David Foster Wallace, *This Is Water: Some Thoughts, Delivered on a Significant Occasion, about Living a Compassionate Life*, Commencement Speech to Kenyon College class of 2005, YouTube video, 22:43, posted by Jamie Sullivan, May 19, 2013, https://www.youtube.com/watch?v=8CrOL-ydFMI.

4. John Shook, "How to Change a Culture: Lessons from NUMMI," *MITSloan Management Review*, Winter 2010, https://sloanreview.mit.edu/article/how-to-change-a-culture-lessons-from-nummi/.

5. David L. Marquet, *Turn the Ship Around!: A True Story of Turning Followers into Leaders* (New York: Penguin, 2012).

6. Simon Sinek, "How Great Leaders Inspire Action," TedTalk, 2009, https://www.ted.com/talks/simon_sinek_how_great_leaders_inspire_action.

7. Simon Sinek, *Start with Why: How Great Leaders Inspire Everyone to Take Action* (New York: Portfolio, 2011).
8. General Stanly McChrystal, *Team of Teams: New Rules of Engagement for a Complex World* (New York: Portfolio, 2015).
9. Michael Abrashoff, *It's Your Ship: Management Techniques from the Best Damn Ship in the Navy* (New York: Grand Central Publishing, 2002).
10. Vijay Govindarajan and Chris Trimble, *The Other Side of Innovation: Solving the Execution Challenge* (Boston, MA: Harvard Business Review, 2010).
11. Andrew Shafer, "There Is No Talent Shortage," Velocity Conference 2013, YouTube video, 38:05, posted by Andrew Clay Shafer, October 28, 2013, https://www.youtube.com/watch?v=P_sWGl7MzhU.
12. Amin Yazdani, "Psychology of Estimation: Faster Is Not Always Better," YouTube video, 28:12, posted by PyGotham 2017, October 21, 2017, https://www.youtube.com/watch?v=SppL54fVNe0.
13. Daniel Kahneman, *Thinking: Fast and Slow* (New York: Farrar, Straus and Giroux, 2013).

Module 7

1. "DOES17 San Francisco—Convergence of Safety Culture and Lean: Lessons from the Leaders," YouTube video, 31:06, posted by IT Revolution, November 30, 3017, https://www.youtube.com/watch?v=CFMJ3V4VakA.

Module 8

1. Jeffrey Snover, "DOES15 – Jeffrey Snover—The Cultural Battle to Remove Windows for Windows Server," YouTube video, 24:34, posted by

DevOps Enterprise Summit, November 5, 2015, https://www.youtube.com/watch?v=3Uvq38XOark.

2. Carlota Perez, *Technology Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages* (Cheltenham, UK: Edward Elgar, 2002).

Biographies

Gene Kim is a multiple-award-winning CTO, researcher, and co-author of *The Phoenix Project*, *The DevOps Handbook*, *Accelerate*, and *The Visible Ops Handbook*. He is founder of IT Revolution and is the founder and host of the DevOps Enterprise Summit conferences. He lives in Portland, Oregon.

John Willis has worked in the IT management industry for more than thirty years. He has authored six IBM Redbooks on enterprise systems management and was the founder and chief architect at Chain Bridge Systems. He lives in Atlanta, Georgia.