EPISODE 466

[SPONSOR MESSAGE]

[0:00:00.3] JM: Our sponsor Datadog is a monitoring platform that helps teams identify, investigate and resolve issues quickly, all in one place. Datadog integrates seamlessly with more than 200 technologies, including AWS, Docker, PagerDuty and Slack.

With powerful dashboards, sophisticated alerts and distributed tracing and APM, Datadog provides deep visibility into your applications and infrastructure. But don't take our word for it. Start a free trial today and Datadog will send you a free t-shirt.

Visit softwareengineeringdaily.com/datadog to get started and get that free t-shirt. Just go to softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:01:05.7] AB: Emil Stolarsky is a production engineer at Shopify. He spends his time thinking about the impact our software has on people around us. Emil, welcome to Software Engineering Daily.

[0:01:17.0] ES: Hi, Adam. Thanks for having me.

[0:01:18.8] AB: You've given some talks about incident response. What is incident response?

[0:01:24.3] ES: Incident response is a field where we look at how systems can fail, both organizational and systems we build and how we can optimize recovering that back to their normal state and everything around that.

That's mitigating a system failure that organizational and figuring out how to organize the human response component that's bringing the system back to running and then doing a retrospective and looking back at the system and seeing what lessons we can learn from the system failing

and making sure that it doesn't fail the same way in the future. Or if it does, that we can minimize the impact it has.

[0:02:06.0] AB: Is incident response made up of several pieces or steps?

[0:02:12.2] ES: Yeah. In my research into incident response, when you're looking into this field, and it's to maybe my naïve surprise, I could say that there is this whole body of work where there is institutes that are going and looking and the term that will be used, there is emergency management.

There, it's broken down into four components. It's broken down into mitigation, preparedness, response – response and recovery. The four components of mitigation is systems will fail, things will break. How do we reduce the risks in making sure we have a safe failure?

An example of this is like on a construction site, you might mark off zones under the crane where people can't walk as the crane is operating, because if the crane breaks for whatever reason, something will drop in that area.

For us, in software that might be something like having bulkheading, or circuit breakers, if say everyone's service is not working. Preparedness is how do we – I think like the human component, so the analogy I would think of intact would be on calls. You don't ever want your system to break, but you assume it will. Figuring out organizationally who is going to be the person who comes in and gets alerted when that breaks, that would be preparedness.

Response is actually fixing it. The service is broken, you deem to bring it back up. Then recovery will be going and looking and doing a retro on it. Recovery sort of you getting back to business as usual or operating. If we're going back to the tech analogy a response will be, you say switch over from a highly available database.

Your first one goes down. Your second one is up. Bringing to the second database in search of the secondary will be the response, then the recovery will be standing up and knew you're highly available database and having a new secondary for the database that's running.

[0:04:18.4] AB: You had mentioned bulkheading. What is bulkheading?

[0:04:22.6] ES: Bulkheading is when you have say a slow external service and you have a bunch of app service calling out to it. Bulkheading is this idea that you don't want all your app servers to be waiting on a response permit, where you will sort of say have – allowing for certain percentage of connections in your applet to be connected to one service.

You assume that if that percentage goes over, or if the number of connections goes over, you can assume that the latency in that service is too high. You'll do a quick failure and you'll just return a failure connection and the service will have to go into a total degraded state.

It comes from the idea of in ships where you have bulkheads in ships, where if you have water leaking into particular portion of the ship you want to isolate the damage. Where you can handle, say a quarter of your ship being in the water or part of your server is being tied up in one service, but you can't handle all of them being tied up, because then you have no more app capacity.

[0:05:23.3] AB: It's mitigation, because you're mitigating a further disaster of some sort.

[0:05:28.9] ES: Right. You're lowering the risk of failure. You're identifying your risk and you're saying, "How can we lower the impact if this risk manifests and we have an incident or issue with it?" That would be mitigation. There you have it.

[0:05:43.7] AB: Mitigation is preventing problems. You had an anecdote about how airlines, or airlines manufacturers tackle mitigation by tracking parts. Could you share that?

[0:05:57.2] ES: Right. In my strange loop talk, I brought up this idea of how every single part in an airplane is tracked meticulously. It's tracked when it was first manufactured, it was put into the aircraft, when it was last serviced, how many flights it's been on and what has been its operational history.

Then after a set amount of time, mechanics will have to go and inspect the part and decide whether or not it can continue flying, or maybe it needs to be fixed, replaced, etc., etc. I'm

thinking well, it does look like in a codex. Often, when we decide systems, we'll build up a system. We'll think a lot about how the systems operates and they will ship it.

But we don't track usage individual, say function calls. It would be interesting of like, imagine every time you deployed code, we started tracking how many calls – there were to every single function. You said after a billion calls this one function, you'll open an issue to go and look and to examine, "Hey, did we design this properly? Is this becoming technical that should we just leave it as is, should we remove it and move to a different mode, should we refactor it?" Of course, if you have a large enough code base that's on your list. But that idea of tracking the age of code is interesting, I think.

[0:07:21.5] AB: I like it's idea. It's interesting. If you're tracking it by the number of calls that are happening to it in production, I guess that's going to lead to a refactoring kind of the code paths that get called the most. I could think of some alternate ways, like where you could say if the code this calls, or is calling this changes, then maybe it should be reviewed?

[0:07:21.5] ES: Also, I think age of code would be very interesting. If you had a heat map of least recently touched code, which wouldn't be too hard to generate by looking at the gate commits of say a code base. It would be interesting to see what's the oldest and what's the newest parts of the code base.

Maybe the oldest part is worth the best, you don't need to type them. But having that looking at code from that perspective could be very interesting. Rather than just ship and then go back and fix it when it's become this big issue of technical debt.

[0:08:17.6] AB: Like look at this code hasn't been looked at in a while. It should be reviewed.

[0:08:22.4] ES: The practices of meticulously tracking parts in the airline industry came from having a series of accidents that were due to part failure. In retrospect and sort of, "Hey, we need to be constantly tracking this to know, because we know that certain parts will fail under these conditions, or will fail after these many uses."

I haven't seen an analogy like that that we could use intact, or the recovery using intact. I felt it could be an interesting approach, and it could lead to some different perspective of how do we prioritize maintenance work on our own code bases?

[0:08:58.0] AB: Should we be tracking the probability of failure across our software components?

[0:09:05.3] ES: One thing we do at Shopify today is we use resiliency matrixes extensively. Resiliency matrix is you'll have on your Y-axis you'll have it greater, and in your Y-axis you'll have every component in an application. Then on your X-axis, you'll have different services across the entire application architecture.

There will be three entries for every service. The first one will be healthy, the second one will be degraded, if they're all be down or complete outage. Then you can track the state of the component in the application to see how they react in that condition. If the database is partially down, can the application still serve something, or is it just go return a complete 500.

If elastic search is down, search what might be down, but you can still complete a check-out. We've been looking in sort of trying to figure out what are the different failure scenarios. In more complex systems, say like aerospace and airlines, in complex industrial process, like in chemical plants, they'll build diagrams out for every single individual component. Then they'll attack different probabilities, or risk factors, maybe that can put in failing.

They'll also map out in these trees, the different relationships and dependencies they have between different components. Maybe a door, a cargo door will fail if two particular bolts will fail. Or maybe it will look at if one of those bolts will fail, then the bolts have their own dependency of trees of what will cause them to fail.

Building out these maps of how can our systems fail, what is the chance of them failing in different scenarios is powerful. It allows us to sort of realize, "Hey, maybe a whole large chunk of our application is dependent on this one component." Then that tells us, "Oh, we have a very high-risk of this component. All our best effort – or it would be best for us to direct all our effort to mitigate that one component and make sure it's a lot more easily into failure."

[0:11:18.7] AB: It helps you pinpoint, I guess like linchpins or very elements that could have cascading failure stories, I guess.

[0:11:30.6] ES: Yeah. In the talk, I talked about probabilistic risk assessment, which is the overarching topic of what are the different ways you can look at a system and figure out the chances of it failing in different scenarios and what the different components rely on each other, etc.

[0:11:49.7] AB: What was the preparedness in emergency response?

[0:11:54.3] ES: Preparedness is how do responders prepare, for lack of a better word, to an incident or failure? In my talk, I talked about the incident command system. The incident command system was developed after a series of forest fires.

In Southern California, the response to them was mismanaged. What ended up happening is the LA City Fire Department and the LA County Fire Department both competed and they had this miscommunication and almost chaos in how they responded to the fire. After the fire was extinguished, people went back and looked at it and they saw that these two organization is not communicating together, substantially exacerbated the size of the fire and the impact of it.

They went off and they developed a system, organizing response to fires. It was called the incident command system. The idea behind incident command system is that during incidents of just failures, you'll have one person who is in charge of responding to the incident. They'll have complete, or almost complete authority on how to respond to it.

They'll delegate and they'll something – like a portion of people need to go respond to this and that structure and placing that structure and having that structure laid out beforehand was very valuable. An interesting analogy for an incident commander would be a composer in an orchestra.

The composer can't play every instrument individually in the orchestra, as well as the musician. Yet without the composer, we wouldn't have the final piece or the final composition won't be as

great. It's this idea of like somebody who's organizing the response is what makes the every individual component, the sum of it much greater than the absolute sum of it.

[0:13:56.5] AB: Does that mean that an incident – so that the person who is the incident response person – what are they in charge of?

[0:14:04.3] ES: I can give you an example of this works at Shopify. At Shopify, we have a dedicated IMOC role, so an incident responder role. That's an on-call rotation of production engineers, so at Shopify our three roles, we call a production engineer. In addition to their normal on-call, they'll sometimes go onto this other on-call instead. What will happen is if an incident is severe enough, the IMOC will come in and they'll be the incident commander.

What that means is their job is to make sure that all the on-call personnel that are necessary to mitigate the issue, or to respond to it are present. They'll facilitate that. If on-calls need to get somebody else to come online, the IMOC will go and make sure that happens.

The IMOC will be the person who's in charge of tracking the incident. If a responder comes online, they will update them on the situation. They'll also be the ones who communicate with stakeholders.

In the past, if an incident wasn't pure enough, there wasn't anybody – it was nobody's job to specifically go and update the status page, or write the status page message, or say inform leadership. IMOC is this formalization event.

It's interesting, when we first rolled this out, you realize that there is in place already. If you think back to times when there is an outage or a severe enough incident, there is one or two people who are managing the process, but they're never elected. They never come and go onto what he's responding. It just naturally happens.

With having a dedicated IMOC role, or dedicated incident commander role, you cannot not only clarify who is going to be doing that role. But then you can also roll out appropriate training and give the best techniques.

One thing we have is we have an IMOC bot, which is a chat apps bot that's integrated in a main chat apps tool, that will help coordinate the actual incident. During incidence, we can add notes that we'll then layer – show up in our RCA docs with this.

The IMOC bot will also send one-on-one Slack messages to certain people with checklists. For example, let's say make sure to update the status page. It will say – make sure to lock deploys. Does this look like a broken code issue, can you roll back? It's been three hours, do you need to swap out your IMOC role right now with somebody else?

All these formalization is very powerful, and putting a term on it and putting the idea on it and then having dedicated people focus on that role has tremendously helped us with reducing our time to recovery, and streamlining the overall response process we have to outages at Shopify.

[SPONSOR MESSAGE]

[0:17:21.2] JM: Dice helps you accelerate your tech career. Whether you're actively looking for a job or you need insights to grow in your current role, Dice has the resources that you need. Dice's mobile app is the fastest and easiest way to get ahead. Search thousands of tech jobs, from software engineering, to UI, to UX, to product management.

Discover your worth with Dice's salary predictor based on your unique skill set. Uncover new opportunities with Dice's new career pathing tool, which can give you insights about the best types of roles to transition to and the skills that you'll need to get there.

Manage your tech career and download the Dice Careers app on Android or iOS today. You can check out dice.com/sedaily and support Software Engineering Daily. That way you can find out more about Dice and their products and their services by going to dice.com/sedaily.

Thanks to Dice for being a continued sponsor. Let's get back to this episode.

[INTERVIEW CONTINUED]

[0:18:37.4] AB: An IMOC is – their job is to not actually address the issue, but coordinate the addressing of the issue?

[0:18:45.2] ES: Right, exactly.

[0:18:47.3] AB: Another thing you led into there was checklists. Could you expand on checklists?

[0:18:52.5] ES: In my research around emergency management incident, I was reading about airlines and the power checklist. I happen to find the story of the B-17 and the origin of checklists in airlines.

The story goes, in the 30s the US Army/Air Force was trying to procure new bombers. All the major airline manufacturers had developed their own prototypes for this competition. Boeing had developed the B-17, which had all these amazing capabilities. It was more resilient to damage, it could fly farther than any of the competing prototypes, it could carry a lot more weight. Much people are really excited about this.

It was also with all those added features, it was also a much more complex plane. They had brought all these prototypes out to a test airfield out in Seattle. On the second test flight for the B-17, just after takeoff the airplane crashed.

During the investigation, they had realized there was a pilot error. There is a particular valve that has to be open just before takeoff and during takeoff, but immediately after has to be closed. The pilots have forgotten to close it.

Then to the investigation, a bunch of the test pilots for the army went off and they started thinking like, “What do we do?” Because these weren’t novice pilots. One of the pilots was the chief test pilot for the army at the time. When they came back, they didn’t introduce or rollout more additional training, instead they came back with this idea of a checklist.

This was the first checklist in airlines and in aviation. The checklist was quite basic. It had sort of, do you – these say three or four steps for just before engine start, or before takeoff, during takeoff, after takeoff, etc., etc.

The reason they rolled it out is because they had this realization that the system was so complex, that you couldn't remember every single component right when you needed it in your brain at all times. They put down the most important steps onto this list that pilots can follow.

They took it all the industry by storm, where now if you think of a profession that uses checklists, pilots will immediately come jump into your head. In a cabin, checklists are they are built into the dashboard with the computer. They also find there's full of them. Whenever there's an incident or an issue on boarding aircraft, the first thing a pilot will do is take out the checklist and start going through the steps.

When you look at it more generally – other initiatives have also started beginning to adopt this; in the military, in medicine. When you look at the before and after of mistakes or failures, or needing time to recovery, with checklist, everything gets substantially better. It's almost malice not to start using a checklist surprisingly.

It was surprising to me, because when you go and you think of a checklist – for me personally, I always thought of a checklist as something that took the thinking out of responding to an issue. If a human is responding to a critical issue why would they – as a critical thinker, why do I need to use a checklist?

The reality it turns out is that humans, while it might be good at solving this complex problems will often forget the basics, or will often forget something that's easily overlooked, but it's really important to the recovery.

The analogy I used in my talk was this idea of automating thinking. In tech and in program we automated everything that's manual and repetitive all the time, because why would we redo it? But in incident response, or with checklists you're doing that, but for your brains almost. Going back to the example of the IMOC bot, if you have an incident in production, lock the place. Don't let new changes go out, unless they're related to the response.

While it seems like a very obvious thing to do, there's going to be those situations where you – it skips your resource that bot skipped your brain or whatever and you forget to lock the place, and somebody deploys new code and it exacerbates the issue.

Checklists are this thing that let us go like, "Okay, there's an incident in production, what do we do first? Lock the place. Second, go down the debug checklist and then you have a second debugging checklist or whatever to start figuring out are we seeing how the database is locating, how is the network edge locating, what's our app server capacity, so on and so on."

When you actually see that there's an issue in a particular component, you can then – that's where you want to save all your thinking and time and focus all your energy on that complex problem and figuring that out, because checklist can't help, or can't always help with those that have problems.

[0:24:14.7] AB: Maybe this is just a very small detail, but why do we want to lock deploys when an incident happens?

[0:24:20.5] ES: We follow that process to lock deploys, just because failure happens from change and systems break when something changes, because before they break they're in a stable state. The idea is that if you lock deploys, you won't be changing anything new, so you have your current – you will be introducing new changes like to change the response.

[0:24:42.5] AB: There's already an incident taking place, I guess so you – you're already in a non-stable state, or –

[0:24:48.2] ES: Right. You don't want to be introducing more changes during that. You want to figure out what change brought you to the point of incident, and then mitigate that, fix it, or remove it.

[0:24:59.3] AB: Makes sense. Checklists are another example of an area where other fields have things to teach us about how to respond to outages.

[0:25:10.3] ES: Yeah.

[0:25:12.1] AB: What can pilot communication teach us about responding to incidents?

[0:25:18.0] ES: Another really interesting thing, like demand on my research into other industries was pre-resource management. There was a story that I happened upon for United Airlines 5173. The story was it was a flight from JFK to Portland. On the approach to Portland, as they were lowering the landing gear, the processor had thumped and the gear down success flight didn't turn up.

They weren't sure if their gear was actually down, so they aborted the landing and they started circling around the airport trying to debug the issue and figuring out what had happened. They did that for about an hour, until they decided to start approaching to land. All their engines began to burn out. They lost power and the airplane crashed just before the runway.

It turned out that the airplane had run out of fuel. When investigators went to look at the flight recording, the flight recorders they had heard how both the first pilot, or the first officer and the flight engineer had warned to the captain that they were running out of fuel, but the captain didn't respond and didn't acknowledge.

They had assumed that he had acknowledged, or hadn't heard the issue, but he just chose not to do anything about it and didn't deal with the problem. Around this time, there was a lot of – there is a series of incidents where a breakdown in communication was one of the core reasons the accident had happened.

Maybe there was a miscommunication between the air traffic controller, or the pilots, maybe the tower and the pilots, between pilots within the cabin where one pilot had identified an issue and brought it up to somebody else, but nobody acknowledges it, or they assumed that the person who brought up the issue will fix it, whatever it might be. This has led to countless – almost countless.

What had happened, almost kind of this access – what happened is the FAA, which is the agency in America that regulates aircraft and aviation, constant where the NASA went and did a workshop to try to figure out how do we rectify these issues.

NASA came out with this idea of crew resource management. Crew resource management is a formalization of best practices and communicating in high-stress situations, where time is almost of the essence. Some of these things, or these are very basic. It will be like clearly indicate who you're talking to, specify the issue you're noticing, specify how you've noticed the issue, so maybe that gauge is broken.

Talk about or mention how you plan to resolve the issue and wait for acknowledgement from the person you're talking to. Even as reciting these, it seems so basic. It seems so obvious. Like of course, of course I'm going to be like, "Hey, captain. I'm seeing this problem. This is why I'm seeing –" Like there's nothing – not necessarily blow your mind, but what the airline should notice is that when they formalize these ideas and when they start to train them and almost like making it second nature for anybody who is a pilot to use these techniques, the results are – you can't debate the results. It works.

In my research, I was listening to talks from pilot's talking about near misses, or accidents they were involved in. In every single one of them, the pilot will talk about how they use crew resource management to more effectively communicate. They might tell their co-pilot to look an issue, or they might know if one person is debugging something, the other one is flying the plane and having that really, really helps.

I was thinking as I was reading through the stuff about accidents that I've been involved in, incidents where – let's say there is maybe a major outage and a bunch of people coming in helping, there will be three or four people who'd be like, "Oh, I think this is broken," and they're four separate things.

Then one of those gets ignored, or gets lost in the noise. Then an hour later, people circle around the back and they realize that one of those was really the issue. That was when you need to be fixing in the very beginning. You go, "Okay, why do we miss it?" It's because we don't have the same structure.

When you go in and looking at the emergency – like emergency method in other industries, sometimes a lot of these process, sometimes a lot of it – like I was going through the NTSB – Sorry, go ahead.

[0:30:13.3] AB: Sorry to interrupt, is it the forced acknowledgment? I could see how that would be valuable. I've actually seen this happen where I think this would help in terms of mitigation where somebody mentions offhand, "Hey, the master database hard drive is almost full." It just rolls – everybody moves on, but that – is that the piece that nobody acknowledged like, "Oh, yeah. That's something important."

[0:30:39.4] ES: Right. Yeah. Forcing that acknowledgment is really powerful and it's also – I've seen oftentimes where somebody will just make a statement, but it's not directed to anyone. Nobody will take ownership for it in that moment. Directly making a statement to somebody could force a conversation around it.

[0:31:01.4] AB: How does that influence how you guys do things at Shopify?

[0:31:05.9] ES: One of the things we're looking at is modifying our on-call training and talking about these ideas and talking about how – what are the best ways to communicate and to point out issues you're seeing. In the United Airlines flight 173, the captain not acknowledging was at a time when the captain was above all in charge of the aircraft and you couldn't sort of challenge them.

With crew resource management, this idea that there is no – there might be high interest of managing the incident response, but there's still like, you want to get rid of the human accountability and social interaction sometimes, where you might be a little nervous to say something because somebody is your superior or whatnot and it's like, get rid of these ideas. You're trying to fix the problem. You're all equals. How can we best do this?

I was mentioning how this feels like process event and obvious. In the emergency management industry, a lot of stuff sometimes is just process. Like I was reading NTSB investigation manuals, and the NTSB is the national transportation board. It's the agency in America that is

responsible for any transportation-related accident figuring out what happened. Then once they figure out what happened, deciding whether or not they need to either issue new regulations or a bulletin, or a issue of here is an advice on how to avoid deaths in the future.

It's an agency of 500 investigators whose job is to investigate and figure out the root cause of accidents. When you're reading the manual, since it's a government issued manual, the first few pages are talking about expensing items. When you expense an item to make sure to keep the receipt.

You're reading this like this is pretty obvious. I don't need to know this. But for other stuff like calling out people or trying to break down this formality of in social settings between people can be – is actually very beneficial.

I think one of the reason the tech industry has been looking more and more into this is because you have to look for these golden nuggets in the rough, or that needle in the haystack where you have to work through – like and sometimes really thick manuals, but then a few of those pieces in there were very valuable.

[0:33:49.2] AB: Yeah. It sounds like you've extracted some great nuggets with checklists, crew resource management. Have you learned anything about root cause analysis from this world?

[0:34:01.7] ES: It's interesting in how – in some regards, the technician is actually better at root causes are figuring out or doing postmortems, by retrospectives then other industries. Other industries have very much of a operator error focused mentality. We'll try to figure out who messed up and then they'll just fire them.

Whereas, I find in the tech industry, we've been a lot better of going what happened, how do we make sure this doesn't happen again. It's important, Dave Zwieback talks about – he is a director of engineering at, I believe it's Pandora. He has a book on postmortems. It's a very interesting book where it's this idea – he gives this example of it tells a story of a technology group in a bank and how they had an incident and somebody was fired for breaking the system and how they think through and discusses it like whether they should have shift by the operator, or they should've fixed the root cause or whatever it might be.

He talks about how really what retrospectives are and postmortems are is trying to figure out what went wrong and controlling for bias. Humans are biased for many different reasons. The only way we can fight those biases and do effective analysis of what went wrong is by having other people point them out.

Some sort of biases that you might have is attribution error, or an attribution bias, where you'll identify sort of the root cause of an incident to a single person. Or the name is escaping me right now.

[SPONSOR MESSAGE]

[0:36:06.3] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we have nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot or a real user. That's why we use Incapsula, to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Incapsula checks that request before it reaches our servers and it filters the bot traffic preventing it from ever reaching us. Botnets and DDoS attacks are not just a threat to podcasts, they can impact your application too. Incapsula can protect API servers and micro-services from responding to unwanted requests.

To try Incapsula for yourself, go to Incapsula.com/2017podcasts and get a free enterprise trial of Incapsula. Incapsula's API gives you control over the security and performance of your application and that's true whether you have a complex micro-services architecture or a Wordpress site, like Software Engineering Daily.

Incapsula has a global network of over 30 data centers that optimize routing and cashier content. The same network of data centers are filtering your content for attackers and they're operating as a CDN and they're speeding up your application, but doing all of these for you and you can try it today for free by going to incapsula.com/2017podcasts and you can get that free

enterprise trial of Incapsula. That's Incapsula.com/2017podcasts. Check it out. Thanks again, Incapsula.

[INTERVIEW CONTINUED]

[0:37:56.0] AB: You're saying that a problem with – how does a bias affect generating a root cause analysis?

[0:38:02.6] ES: An example would be if you think of an incident and you build a linear timeline and this idea of a linear timeline is also partially broken. But suppose for now we'll have a linear timeline, there could be a point where an operator – so a programmer, operations engineer, production engineer makes a change and the system breaks.

When you look at it in that context, or in that link it looks like that person made that decision. They made a decision to shift broken code, they made a decision to delete the wrong database. I literally think back to the outage that GitHub had a while back, where an operator there had logged in to the wrong database machine to do maintenance and ended up deleting the wrong database.

When you in-app postmortem and it's written, so so and so logged in and deleted master database or primary database. It looks like they had logged in, they knew they were on the primary database and they decide if they delete it. But that's not the case, right? It's people think they're making the right decision up until the moment they make a mistake.

We have all these different biases when we look back, and it's important to try to build tooling and use different processes to control for them and to try to lower the chance of having those biases come into our decisions going forward. These postmortems retrospectives are super valuable. You don't want to repeat the same mistake. If you can figure out what is the core reason, or root cause that's causing multiple other issues throughout your system, that's invaluable, but it's important we get there the right way.

[0:40:01.1] AB: You have to make them not a – trying to find who is at fault, but more look in terms of – in terms of process, or how you would change processes. Is that the idea?

[0:40:13.5] ES: For instance, at Shopify we expose a lot of tooling around say flushing our caching system. An example might be – I don't think we ever had this issue, but suppose something accidentally flushes the caches without intending to. One approach in the retrospective can be why did you flush the caching system? Do you cause issues, I suppose.

Another approach could be, why was it so easy for you to flush the caching system without the intention to do it? Why could somebody make a mistake of being on the wrong database node? Why was it easy to ship broken code? Maybe the conclusion is that because in order to have "perfect code," or have such a test week that like it's that particular valve could be super low, the cost of it is too high.

Maybe it takes too much time to run a test and so the tradeoffs we say, "Okay, this is a risk we decided to take." But having trust in the people in your organization is very important in figuring out the systems around them and help ensuring that they have the right tools to not cause issues is where postmortem should focus on.

[0:41:34.1] AB: I've seen where a way that that's handled is with the five whys, like Dave deleted the database, but why. Is that a useful way to get to these root causes?

[0:41:49.4] ES: That's one of the ways. There's many ways out there. I'm still going out and trying to categorize and each one has its own bias. One interesting one that I really sort of – I like the idea of was a console factor tree. This is one used by NASA, where they'll build out a tree, all the different components involved, different events and how different components failed.

On each of those events or components will have leaps or nodes that trees under them that will talk about how they failed or how they gotten to that, say the history above. I like it, because you get to see the reality that an incident very often is multiple things going out on parallel. Each thing has its own independent timeline. In a console factor tree you could lay all that out.

But then another thing that happens with the console factory tree is that very often when you go far back enough, you'll hit an organizational component, and then that will be at the bottom of

the tree. One of them might be like, there weren't enough engineers to fix the technical depth, let's say in an example. You could have a bias in that sense.

You're never going to find an approach to your problem with – to a postmortem without any biases. What you should be doing and aiming to do is looking at the different tools you can use, the biases that will come with those and then keeping an eye on making sure you don't succumb entirely to those, or that you're aware of them in making sure that you're accounting for them in your decisions and your conclusions.

[0:43:38.1] AB: Once you've generated your RCA and try different methods to eliminate biases from it, is there end results, or what's the goal? Actually let me rephrase this whole question. Should we as an industry be tracking our CAs in some global-cross industry manner?

[0:44:03.2] ES: Okay. Yes. A million times, yes. If the rest of this podcast would just be me shouting yes, then maybe we should do that. Another story that I came across with my research is I find it very exciting. It's the aviation safety reporting system.

The director of the aviation safety board was getting a speech and he talked about how – the reality is that every single airline has a huge database of all the accidents or near misses and experience. An airlines legally require to report accidents that have occurred, but if it's in your miss, they don't have to.

The director was saying how we're not capitalizing on these lessons we're learning, because only the lessons are staying silent. They're not being sure to cross the entire industry. What came out of that is a database was started where every pilot could submit an anonymous report of an accident that had occurred or a near miss that occurred.

The database is managed by a mutual third party. In this case, it was NASA. Anybody in the industry, even you can go and look at this and read the reports. You can see what have happened, what was the environment that had happened in. In addition, the FAA like I said yes so much, that you actually get legal immunity I believe for up to five or 10 years after an accident has occurred if you submit the report.

If you submitted a report and you did something that was wrong or illegal, but you talked about it and you let other people learn lessons from that mistake, you can't be faulted for that. That's a really exciting idea, because well, we have our own different flavors of systems we all build. They're all very similar. If you take a web application that's running rails, is using MySQL as its main databases, choosing elastic search-by-search, it's using Redis for its job queue system, I bet there has been numerous incidents in every one of those companies that are very similar.

The first company I talked about this mistake it did, maybe the replication setup wasn't optimal. Maybe a particular setting has a different symptom that you don't expect as a problem, until something else completely different in this architecture breaks.

All those other companies wouldn't have had to pay the same price and figure out that lesson on their own. Imagine if there is a third party database, or a database that didn't have any goals of profit, but just to better our industry. Where every company can submit their service disruption reports, their retrospectives, talk about the lessons they've learned, and anybody else can go and read about that.

Sure, we'll have anonymize say the timestamps in it, we'll have to anonymize maybe some of the specifics, but the ideas are largely the most important part. You can even sort of, I can imagine something for us would be in the tech industry, this organization then would go off and be able to develop best practice guides, right?

If you go and look at all the different failure scenarios of say engine active production, you can say this is one of the optimal ways to run engine active production, because it comes for all these different very common incidents.

[0:47:45.6] AB: This is a great idea. I can imagine yeah, an engine X consulting company putting out some white paper where they look through all the incidents and their advice in everybody about best practices. Why anonymous?

[0:48:01.7] ES: So nobody can go out and get blamed, I guess. Anonymity provides protection, where we don't necessarily care who was involved in the accident, we care what had happened. Because the individual themselves – like you can swap with a different and if the process and

systems in place will cause people to create – to make mistakes, then it doesn't matter who that person was. What matters is what happened and what made that happened and what the repercussions of that were, so we can go and mitigate them.

[0:48:36.4] AB: Yeah, it's a great idea. You had mentioned earlier SREs. That's a Google role, I believe. How is the SRE role influenced how you guys do things at Shopify?

[0:48:49.4] ES: The SRE role is Google's sort of term for it. Production engineer is really a synonym. It adopts the SRE mindset, but the difference is largely only in the name.

[0:49:01.6] AB: Okay. Could you expand on that? What does the role encompass?

[0:49:04.5] ES: The idea with traditionally in companies there would be a developer and an operator role, or an operations engineer. The developers write the software of the service that will run in production, and then will throw it over a fence to the operation engineers who will deploy that service and manage it and maintain it.

If there's an outage, the operation engineers are going to be the ones who will try to fix the service. Not the developers who wrote the code. The idea with the SRE role is instead of having this divide, the operations engineers built the tools and systems that developers can then use to run their own software.

Imagine if you had your internal Heroku. Developers will write their code and then they push to Heroku, and then they can look at – they can monitor their own application, they can figure out if the application needs more resources in particular type. The SREs in an organization would build that Heroku almost internally and the actual manifestation of what that looks like in reality is different, but conceptually it's very similar.

[0:50:10.4] AB: the value of having such a new position is?

[0:50:16.4] ES: One way to think of it is as your service scales up, the number of machines you're managing or dealing with also grows. The number of operation engineers you need as your service grows scales linearly with that. With estimate and SRE type role, since the focus is

on its – you can almost think of it as developers with strong systems understanding who are automating a lot of the manual process you would have in a traditional operations role. They'll scale logarithmically.

You don't need a massive – you need a substantially smaller organization to be able to manage a large service. Then also forces much healthier ideas around interacting with any infrastructure, so there's this idea of herds versus cattle. Where before, if you're manually managing your system, you would treat each computer or server as a pet. You would manually fix it, you would come in and you try to figure what the problem is, you would almost – It's very one-on-one.

With guest remodel, they did that you want to automate away everything. You want to automate away all these toil. You'll treat your computers like cattle, where they'll all be treated in the same. There won't be any special stuff like why computer has Y configuration, another one has a different one. If a computer is misbehaving, you can just wipe it and reinstall the same configuration that you had in all the other computers and treat it not as an individual, but as a part of a herd.

[0:51:52.3] AB: The difference being that each pet has a name?

[0:51:56.4] ES: Yeah.

[0:51:57.6] AB: Is unique to you, or every cattle is the same, I guess?

[0:52:02.7] ES: Yeah, exactly. An example would be if you have cache 1, cache 2, cache 3, cache N, like that's the treating them as cattle. But if you have like cache – rail's cache, cache page cache, cache whatever, your each cache server is unique in the infrastructure and is especially different and that's not great for long-term manageability.

[0:52:28.7] AB: I want to be conscious of your time. Before we go, what do you find to be the problem with the Facebook model of old move fast and break things?

[0:52:40.5] ES: I think two things; one, in the tech industry, traditionally – or not traditionally, but in the pat and when we were younger, the services that we built they're impacting the people around us and their impact on society was much smaller in scope. People's lives didn't rely on this thing called the internet.

Today, when our services fail there is a problem, the consequences can be terrifying. People can't travel, banking grinds to a halt, or 911 response services can't work anymore, so on and so on, the list is countless. The terrifying thing is that it's growing – it's just growing by the day. We're constantly modernizing and connecting all these different things, like before we're analog and now they're becoming digital.

We need to as an industry start to appreciate the responsibility. We have people who are technologists and approach our service – the things we build and the systems we built maybe not with the extreme of managing a nuclear reactor, but there's a lot of lessons out there that we can learn to make sure that our systems are more stable and are built with that understanding of their importance of standing up and available.

Move fast and break things is indicates to me this old idea. A time before when if things broke, it's fine. We need to move to something where it's more break – we can't just let other people pay the price of our systems breaking.

[0:54:30.9] AB: I think that's a great thought and it's a great place to leave this with. Emil, thanks so much for your time and all your great insights.

[0:54:39.3] ES: Thank you. I had a ton of fun.

[END OF INTERVIEW]

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery are available. Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[END]