MOVE NOW
GET RESULTS

cycloid

Prime your org for optimum DevOps take-up

TABLE OF CONTENTS

PROLOGUE

DevOps is going to allow you a faster, more efficient, more modern organization. It's a force-multiplier, helping everybody and everything it touches to optimize, refine, and speed their work. By leveraging it to automate what you can, your tech team will be left with the time and mental space to concentrate on what truly matters - the transformation of your company.

Great! So...where do you start?

It's a huge task. As you know, there's more to DevOps than "*automate it, and it will transform your company*". Apart from the IT automation tools and solutions available, you need the biggest piece of the pie - the automation mindset. Solutions and tools can be tried, tested, demoed, and researched, but it's the right mindset that will make or break your journey to DevOps nirvana.

What are you waiting for? There are behaviors you can model, changes you can encourage, and information that you can share that will pave the way for a team that's ready and willing to get the very most out of automation. We're going to help you absorb what we call "DevOps thinking" - a mindset that echoes the main, most theoretical principles of DevOps - and the one thing that cannot be bought online.

So read on, find out what "DevOps thinking" means and put it into practice today - your automation-first team of tomorrow will thank you.

DevOps thinking and automation - what's the link?

Some say that "automate everything" is the unofficial (or official?) DevOps motto. Although some people say it facetiously, it really is the ultimate goal - to automate everything you can in order to free up your excellent brain to tackle more complex problems.

In DevOps, automation can be applied all the way through the SDLC (software development lifecycle), from generating code to pushing to production and after. In fact, there's barely an aspect of software development that can't be upgraded with a little automation.

If automation is your goal, it's undoubtedly to speed up and optimize your production process and to free your devs from drudgery. If that's so, your goals and DevOps' goals are one and the same, so you might as well leverage the extensive work that's already been done and get a head start on your automation journey.

CHAPTER 1

CREATE A SHARED, COLLABORATIVE VISION

Your first step in the move towards automation nirvana is to create a shared, collaborative vision of what nirvana looks like. You see the benefits clearly, but if your devs - or your non-tech colleagues - don't, you'll have a hard time convincing them along for the ride.

Of all the changes you're going to make, this one has the lowest barrier to entry. Your task is modelling, encouraging, and making changes that seem innocuous and in the best interests of any right-thinking person, not just one who loves DevOps.

> **“THERE IS VIRTUALLY NO ENVIRONMENT IN WHICH TEAMS—IF DONE RIGHT—CAN’T HAVE A MEASURABLE IMPACT ON THE PERFORMANCE OF AN ORGANIZATION.**

Jon Katzenbach, The Wisdom of Teams

Breaking down the concept of a “shared vision”, you’ll actually be:

- **Improving teamwork**
- **Bolstering collaboration**
- **Encouraging autonomy**
- **Breaking down silos**
- **Ending “throw it over the wall” anti-patterns**

Now, you may be thinking - although these are all positive changes, they’re completely unrelated to pushing forward IT automation, right?

Wrong.

In order to automate a large portion of the tasks that contribute to your product, you’ll have to employ DevOps thinking. To employ DevOps thinking, your team must be able to truly work collaboratively. To roll out automation, you must give your team authority to make significant decisions without approval. To trust your team to take these decisions, they must be highly tuned in to the needs of the company.

See how it’s all linked?

So, now we know why this is important and understand that it’s something we can do right now, let’s take a look at how you can actually increase collaboration, warn you of some pitfalls, and show you where to go next.

From “my” job to “our” company

Firstly, you must create a shared vision for your team. Moving the mindset from “my department” to “our company” and “my team” to “our product” is essential, but not a quick fix. It’s something that you’ll need to work on constantly and patiently, but it’s worth it - it’s how you’ll encourage people to take autonomous decisions that actually benefit the company, and not just themselves.

Familiar with Reddit? Then think of it in terms like ELI5[1] - if you can't explain your vision for your company to a 5-year-old, you don't really understand it yourself - go back to the classroom until you do!

The first and easiest move is to model the behavior you want to see. Start speaking in terms of "our" whenever possible. Reiterate and act out what you want to see, and keep talking about the teamwork nirvana you want to reach.

Once you've focussed on helping people to see the product as a team responsibility, take some time to understand and demonstrate how exactly they, as individuals, are contributing. Although we want fluid and collaborative teams, humans still need to understand their specific role in success.

Allow team members to see progress towards common goals and make sure they understand (and others can see) where their efforts have helped. OKRs (objectives and key results) can help this process.

OKRs and the collaborative team: ***OKRs often prove more helpful than traditional KPIs or other goals as they highlight progress towards a common goal, rather an individual or team-specific goal. When using OKRs, you define a clear and common objective and 3 - 5 key results - specific measures that will indicate how close the team is to achieving that objective. When done right, this objective will be common to the whole company - not just any one team. The market is full of tools that can help teams implement OKRs and information that will bring you closer to understanding them[2].***

[1] https://www.dictionary.com/e/slang/eli5/#:~:text=ELI5%20is%20short%20for%20%22Explain,a%20complicated%20question%20or%20problem.

[2] https://gtmhub.com/blog/okrs-for-the-engineering-team/

Common goals in practice

It's really important that you make it your business to ensure the roadmap to your collaborative goals is crystal clear and regularly reiterated. Recognize and reward good collaborative behavior and help everyone feel good - and successful - about their new approach to work.

It's worth pointing out that even if your organization was not previously very silo-ed, it will still benefit from a renewed and fresh focus on collaboration.

Remember that great collaboration is, in essence, great teamwork, about which there is a lot of high-quality reference material that you may already have within your company library or easily accessible with your L&D (learning and development) budget. Even more, share this material with the whole team, allowing everyone - not just managers - to benefit from the wisdom.

Further ways to encourage collaboration

Where possible, facilitate cross-functional training. This has many benefits. First, it can simply help teamwork and collaboration, as exposure breaks down barriers and creates new friendships. It also reduces the famous bus factor (box below). Finally, when multiple and different people consider one issue or area, it will foster diversity of thought, which very often yields approaches to problems that may not have previously been considered.

What's a bus factor?

The bus factor is the name given to the (good) idea of having more than one person who can do a given job. If only one person knows how to carry out a specific task, their absence (if they got knocked over by a bus, for example) would cause everything that revolved around that task to grind to a halt.

If more than one person knows how to do that task, then the absence of just one person will not throw the whole process into chaos. The bus factor is the number of people who would need to be removed from the equation before productivity is impacted. Ideally, the higher the number, the better. You might lose Bob to an errant cross-city express, but you'd be very unlikely to lose Bob, Martha, AND José!

Ending silos and the "throw it over the wall" mentality

Silos and "over the wall thinking" have become buzzwords, but they both hide a damaging way of operating that will ruin or prevent collaboration. Leave either unchecked for too long and people become disconnected from the whole. Team members begin to grow resentful of each other - one thinks "it's not my problem, it must be theirs" and vice versa, leading to a negative loop and rarely any fixes.

In the silo, each role is assigned to a different individual and there's no view of the bigger picture. Individuals see the information that is right in front of them, but can't see what that information plus everyone else's piece of information adds up to.

Why are silos bad?

If you're a manager, you've probably heard of - and been taught to fear - the silo. A silo happens where a specific team or department considers their job to start and end with a specific task - their job. They view the job solely in terms of how they interact with it, paying no heed to the parts that come before or after, or how they fit into the bigger picture. This approach to development slows things down, discourages collaboration, and is totally anti-DevOps.

Likewise, in “over the wall” (or “ping pong”) cultures, every time there’s a question or problem, a lack of responsibility and collaboration means it’s thrown to another team or person to solve - nobody works together to find a fix (information sharing) or to change the process so that a similar problem doesn’t happen again in the future.

This can lead to disrespect for the person/s on the other side of the wall - they’re seen as the ones who cause or are unable to fix problems, which results in a lot of resentful back and forth in an attempt to find a fix, rid yourself of blame, or simply progress to the next stage.

Where there’s a silo, you’ll often find over the wall thinking, and vice versa. An environment like this doesn’t encourage people to learn or take ownership - in other words, to be autonomous.

So what’s the fix?

There’s no one fix or solution - it’s more a case of seeing a reduction in this kind of behavior with better teamwork, increased autonomy (responsibility for “the code” rather than “my specific role in the code”) and plenty of cross-functional modeling and encouragement from you.

Start weeding this kind of thinking out with early collaboration and iterative processes. When everyone is involved in early and collaborative discussion, it’s hard for *your/my* thinking to even get a foothold. This is much easier with smaller units of change - if it’s any bigger, the time needed for change will increase. With so much “hands off” time for much of the team, it makes it more likely that your/my thinking has a chance to creep in.

Increased authority, which we'll talk about later, will also help to eradicate ping pong thinking. Consider making developers responsible for their own things in production - when they're responsible for them, they'll be much less likely to try to pawn it off onto someone else. When a dev is responsible for the code all the way through the process, papa duck behavior will set in and they will accompany their code all the way to its final destination, rather than tossing it over to testing and moving on with their lives.

Sure, you might face some pushback from ops over access to the pipeline, but with some clever thinking, any self-respecting op can set up railroad pipelines, limits, and safeguards that will ensure that any access is safe.

How does breaking silos down help?

By breaking down silos, you'll give your team members a chance to truly work together. When everyone is aware of and involved in the different stages of the software development lifecycle, collaboration improves and things get faster. When team members are aware of other parts of the process, they can make better and more informed decisions, which will speed up delivery and make processes more agile. Finally, a truly de-siloed team has a much improved bus factor, something you, as a manager, can be very grateful for!

Autonomy and good collaboration don't completely protect from the silo/over the wall mentality, but it is something that's often seen in isolated teams on autopilot, who tend not to display these traits.

What's blocking collaboration?

Although collaboration is great in theory, it's not always easy in practice. Remember the "Ikea effect" if you're feeling despondent - it's the idea that "labour alone can be sufficient to induce greater liking for the fruits of one's labour". In other words, people feel better about things they've helped create, and tech teams are no exception. Find ways to remind and congratulate devs on their collaborative good intentions in a project, no matter how minor.

There are a few steps you can take to ensure that collaboration happens more easily. For a start, encourage those in hiring positions to actively recruit good team players who naturally look outside their immediate circle for interaction. Get into the habit of asking "is this person good, or good for the team?".

Put some work into identifying physical, emotional, and informational blocks between people and teams. Some areas that you can begin to look at are:

- **Physical blocks** - in a physical environment, this is pretty obvious. What's keeping people from collaborating? A lack of meeting rooms? Vast distances between colleagues? An office environment that doesn't encourage casual conversation?

- **Remote blocks** - In a remote environment, this is a little trickier. The barriers might not be so obvious, but there are many great resources[4] you can use to identify them regardless.

Remote teams and remote collaboration

If you thought that your team could improve communication when they were all in the same office, welcome to the new reality of communication problems when everyone is dozens - sometimes thousands - of kilometers apart.

[3]https://blog.hubspot.com/marketing/remote-miscommunication

Communication in remote teams is a whole area of research that is currently of hot interest to almost every manager around, especially in the field of tech. The problem is magnified if you are dealing with newly-remote workers, as you haven't necessarily had the chance to hire for skills that are of particular benefit for remote teams, like communication skills and proactivity.

If you're lucky, the remote hiring problem will have been noticed at the top levels of the company and you might be able to hijack some pretty intensive resources or options. If you'll actually have your hand in new hires for the tech team, look at the problem as one of two parts. Firstly, you'll need to make allowances for the actual technical part of the interview. It's pretty straightforward but will likely benefit from some streamlining and testing.

Secondly, you'll have to find people who are a good remote fit - culturally speaking. Soft skills will be of key importance here; things like communication skills and proactiveness. If your developer doesn't natively speak your language of business, it's essential that you check verbal and written skills, as sometimes one will seriously outperform the other.

You can also check for past evidence of collaboration, perhaps using methodologies that emphasize very frequent communication, like Agile or simply asking for past experience, problems, or concerns.

FInally, if the person seems a good fit, make sure you clearly explain what "life" on your team would look like and how, although remote, communication and collaboration is key.

- **Cultural blocks** - Some people will pishposh the idea of cultural problems, while others will nod their head sagely. Cultural and language issues can manifest in many different ways, from reluctance to interact with certain types of people or job roles, to a lack of language skills keeping people quiet out of fear or embarrassment. This will take investigation and tact to pinpoint and we strongly recommend bringing HR into the equation.

Bear in mind that we're not just talking about the types of cultural problems that diverse geographical backgrounds can bring - some people bring with them the expectations of a given professional culture, which can also prevent them from collaborating effectively with members of a team.

- **Political blocks -** yuck! Office politics makes you think of Mad Men, right? Some companies are very determined in their efforts to eliminate office politics, but to varying degrees of success. If you think they're still having a negative impact on your teams' ability to collaborate, you'll have a tougher job ahead of you. Unfortunately, or maybe fortunately, the recognized cure for office politics[4] that create a toxic culture are the very same ones that you'll use to maximize collaboration - clear goals, better communication, and meticulously rewarding the right people for the right reasons.

- **Hierarchical blocks** - are people afraid to collaborate with people they see as senior to them? Are senior people dismissive of more junior people's requests? Both will hinder communication and will benefit from the help of HR to fix. In the meantime, however, model the behavior you want to see and act as you wish others would.

  Finally, frequent, informal, and optional social events will also help create better relationships between teams.

When shared visions don't work as expected

Every so often, you'll find a team that has been given the shared vision speech before, and isn't falling for it this time. Especially dangerous is where teams have been asked to collaborate or be autonomous, but then unfairly blamed or held responsible for failures. If this has happened and your team is wary of change, you have a harder job ahead of you, but not impossible.

Start basic. You simply need to show - through repeated and dogged modelling - that things are different now. Keep your promises, don't betray confidences, say what you mean, and own up to your mistakes. Once you have established personal trust, move onto the rather more intimidating task of convincing others to do the same.

[4] https://www.americanexpress.com/en-us/business/trends-and-insights/articles/power-tripping-5-ways-to-help-manage-office-politics-and-infighting/

Fear is a huge destroyer of trust. You must make it your mission to eliminate all and any situations that make people afraid of well, anything, in a professional situation. While you can't do much for heights or spiders, you can help build a culture where it is completely unacceptable to humiliate, badger, or intimidate a colleague.

Need convincing? Research from Google[5] shows that psychological safety is one of 5 key things found in high-performing teams. And if Google has called them high-performing well, you can be pretty sure it's worth learning from them.

Facilitating autonomy

Your next stop on the way to an automation-friendly, DevOps-first organization is to foster autonomy and independence wherever possible. This is a mindset question and people will follow the example you set.

Now, we're going to be very optimistic and assume that you have spent some time maximizing your team's collaboration, mutual admiration, and cohesiveness. You have, right?

Now that everyone is working together and thinking about their role in the furthering of the company, rather than their role in the weeds of whatever team they're on, it's time to encourage everyone to take more autonomous decisions.

There are a number of steps you can take to make this happen right now, but it will mean that you, as a manager, need to breathe, eat, and sleep the idea of facilitating autonomy. To truly do this you need to fully understand what a vision of "autonomous thought" will really mean in your organization - how will teams operate? What needs approval? Who will give that approval? Who is responsible if things go wrong?

[5] https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/

> “THE TENSION BETWEEN EMPLOYEE EMPOWERMENT AND OPERATIONAL DISCIPLINE - STOP THINKING ABOUT THEM AS ZERO-SUM EXTREMES[...]

Ranjay Gulati, Professor of Bus. Admin. at Harvard Business School

Autonomous thought and decision making

Authority and responsibility are an essential pair - you cannot have one without the other. If autonomy is given to your team, you must also make them responsible for the consequences of their decisions, unless there is a very unusual and compelling exception. Responsibility without authority is crushingly unfair, and authority without responsibility is pure chaos!

Authority doesn't mean you can never again create guidelines for your teams - in fact, they can actually improve autonomous decision-making by providing reassurance and guidance, which makes people more confident in their decisions.

If there are concerns among your team members about being more autonomous, listen to them, even if that means clashing with management. It will maintain the trust you've been working hard to build and ultimately empower people. You may also learn something in the process.

As we've already mentioned, bullies should never be tolerated in a professional environment. That said, we all know stronger personalities who, under different pressures, can have bullying tendencies. Find these personalities and tame them or weed them out - bullies will impede others from making confident and decisive decisions.

There are many accessible ways of building up to overall decision-making authority, starting very small. Flexitime, remote working, and flexible vacation policy can all help people build their autonomous decision muscles and can have a huge positive influence on overall wellbeing.

Likewise, if there are certain areas where you can't offer autonomy, take the time to truthfully explain why. Your team will be grateful for the honesty, have an opportunity to learn more about decision-making in the company, and it will help bolster their trust in your leadership.

Once your teams seem comfortable with the idea and you have a rough idea of how it's going to play out, speak to those in the know about what would make a good starting point for autonomy in the SDLC and build it from there.

SECTION SUMMARY

- DevOps thinking will help prepare your team for automation
- DevOps thinking requires collaboration, teamwork, and autonomous action
- Several steps will help foster this:
    - Improving teamwork
    - Breaking down silos
    - Ending over the wall thinking
    - Removing practical barriers to teamwork
- Troubleshooting common problems
- Encourage autonomous thought and action

RECOMMENDED FURTHER READING

Rework — Jason Fried & David Heinemeier Hansson

Herding Tigers: Be The Leader Creative People Need — Todd Henry

Compassionate Management — Rena DeLevie

The Manager's Path — Camille Fournier

Leaders Eat Last: Why Some Teams Pull Together and Others Don't — Simon Sinek

Driven by Difference: How Great Companies Fuel Innovation Through Diversity — David Livermore

Reinventing Organizations — Frederic Laloux & Etienne Appert

Effective DevOps — Jennifer Davis and Ryn Daniels

The Wisdom of Teams — R. Katzenbach, Douglas K. Smith

ARTICLES MENTIONED

https://hbr.org/2018/05/structure-thats-not-stifling

https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/

https://www.americanexpress.com/en-us/business/trends-and-insights/articles/power-tripping-5-ways-to-help-manage-office-politics-and-infighting/

https://blog.hubspot.com/marketing/remote-miscommunication

https://www.qut.edu.au/business/insights/the-ikea-effect-how-we-value-the-fruits-of-our-labour-over-instant-gratification

https://gtmhub.com/blog/okrs-for-the-engineering-team/

CHAPTER 2

BUILDING A CULTURE OF SAFETY

Now that your tech team is working in beautiful harmony, has banished the word “silo” from their vocabulary, and is taking solid autonomous decisions, it’s time to make sure that while they’re flying free and decisive, the chances of them making any catastrophic mistakes is greatly reduced.

You’re going to do that by building a safety culture.

We’re not interested in the excuses for why this isn’t currently a thing in your organization, but from now on, it will be. Remember, building a culture of safety doesn’t involve anything new or different - everyone in the SDLC knows what basic safety is and how to do it - what you need to ensure is that it is actually done.

So, how should you start? Simple.

- Make backups default
- Ensure redundancy
- Use version control
- Build observability into deployed services

Backups

There are 3 golden rules to backups:

- Have them
- Do them regularly
- Make sure you can restore them

To be truly safe (and to adhere to best practice), you should store your backups offsite. This can be completely offsite, or a mixture of offsite and onsite, but at least *some* of your data should be stored off-premises.

Remember that mirroring and replicas, although they have their place, are not backing up. Why not? Mirroring and replicas...well, replicate things, good or bad. This means that if any accidental destructive changes or corruption takes place, they may well get mirrored or, worse, replicated too!

Once you've decided on what you'll back up to, make sure you do it frequently, and always make sure you're backing up more than one copy. How frequently? Well, that depends on how critical the data is. If it's of huge importance, you might need to back up multiple times a day but if it's a little more low-key, once a week might be plenty. Talk to your devs to get more insight.

Can I backup on GitHub?

Well, we all know it and love it, and it's ok for some things, but not others. If all you need to back up is code, it might work but if you've got any binary data it will be way too inefficient. The algorithms Git uses under the hood are inefficient for binary data and, anyway, it changes too much. Once you've decided on where you're backing up to, make sure you do it frequently, and always make sure you're backing up more than one copy.

Ensure redundancy

Redundancy is a way of preparing for the worst-case scenario by making allowances for the probable eventual failure of parts of your system.

Whether it's your servers or your data, make sure one failure doesn't become catastrophic by ensuring there's always an extra to take over. Whether you're talking about hardware, servers, networks, or even internet redundancy, you'll need to make sure that you're balancing benefits with cost (these "extras" are pricey!) and work with those in the know to find the balance that's right for you and the way your team works.

Redundancy vs Resiliency vs Redundancy

Redundancy is a term that gets mixed up in two separate ways. Firstly, people have a tendency to confuse redundancy and resiliency, which, although related and linked, are not the same.

Secondly, people are confused by the fact that redundancy is sometimes something that people want to introduce and, other times, something people seem to be trying to get rid of - what gives?

Redundancy vs resiliency - in very simple terms, redundancy is having more than one of something so that in the event that thing breaks down, there is another one to take its place. Resilience is a quality of a system - it's the ability to absorb a breakdown (even of a redundancy) and maintain acceptable service levels. In order to have a resilient system, you need to have adequate redundancy in place.

Redundancy vs...redundancy - ok, so there's another problem when it comes to redundancy; it's also used to talk about something people would rather not have in their toolchain, rather than something they're looking to build in. This kind of redundancy is when you have tools, scripts or methods that are, in fact, redundant. In other words, you have more than one tool to do the same job or even multiple tools all doing the same thing. This kind of redundancy isn't desirable as it's inefficient - do a tooling audit to weed it out!

Distributed version control

Distributed versioning is what is going to make sure that you work the safety angle from both ends. Regular, reliable backups work to prevent catastrophic data losses, but distributed version control prevents more localized, but still crucial potential losses.

The most likely way you will do this is by using Git, but there are plenty of alternatives. The main idea, however, is that if you have access to versioned iterations of your code, if one version is corrupted or a bad change is introduced, you can simply revert to the previous version and not have to start all over again.

There are other benefits too. If your repos were to go down, for example, your team would still be able to work locally, as long as team members had an up-to-date repo checked out on their machine. Likewise, if the central repo gets corrupted or destroyed in some way (eek!), you'd be able to restore it through the same checked out copies - as long as you keep them up to date of course.

Building observability into deployed services

We'll talk about observability in more detail in both Monitoring Basics and Observability, but take a second to understand why it's important to build it into your systems from the get-go.

As time goes on and you get your monitoring sorted out, you'll have lots of tools and dashboards giving you information, metrics, and data - sometimes by the yard. Observability is the ability to use this data to find answers to questions you didn't know needed to be asked.

In other words, you can have as much monitoring as you like, but if you can't use it to troubleshoot the behaviour of your systems, it is useless. Remember this as you add tools and steps to your culture of safety.

Observability is especially important in organizations that use microservices, because a single request from an end user can end up hitting multiple services, and those services can interact in strange and unpredictable ways.

Monitoring and troubleshooting

Now that you've hopefully taken as many proactive protections against disaster as possible (backups, redundancy, version control), it's time to look at what you can do to anticipate and prevent disaster before it actually happens.

You and your devs are going to do this by ensuring that monitoring becomes a smooth, fluid, and reflexive process that is so well built into your systems that it is entirely second nature. None of this, like everything else about a team that's ready for automation, will happen by chance, so it's something you need to oversee now.

Autonomy in troubleshooting

No matter what failsafes you have organized, sometimes, things are going to go wrong. Whether that's in development or once the app is in production, you need to ensure that team members can safely see and manipulate the services that relate to their code and for which they are responsible in production. On an everyday basis, this means simply ensuring that devs have access to everything they need - if credentials are centrally managed, this will be easy to assess and check.

In other circumstances, you might need to bring out the heavy hitters.

For example, consider letting on-call team members safely escalate privilege to allow them more flexibility to solve problems. This might involve giving them access to databases they can't normally access, or logs they don't normally see. True, there's probably a good reason they can't normally see them but with strong auditing, you can ensure that they have the autonomy they need to do their job, but that the elevated privilege remains squarely within compliance.

Monitoring basics

Helping your team nail monitoring will be one of the best things you can do for them. For a second, let's imagine that all of your efforts towards automation pay off, and your company sees a massive boost in growth. If your company grows, so will your tech stack, and if your tech stack grows, so will your tech problems! It's just a fact of life, but if you have your monitoring locked down, nothing your app can throw at you will rock the boat.

You'll need the help and input of your devs to really nail monitoring, but if you're keeping an eye on the big picture, these tips will help keep you focussed and on the right track.

- **Remember the monitoring-metrics-alerting trio. No monitoring is complete without the other two being in place, so always think of them as a whole.**

- **Decide early what you need to measure to make sure the applications are working correctly/incorrectly (aka SLIs).**

- **Ensure that your services can be monitored for both correct and incorrect behavior - in other words, not just when something goes wrong!**
    - Define incorrect behavior key metric thresholds (service level objectives) early on. Hopefully, you'll have a site reliability engineer who can help you in this task, specifically with regard to ensuring the alerts that it will inevitably generate are actionable and don't create unnecessary alert noise.
- **Make sure your logging tools can log and display the information that you need to know - and know that logging and display are not necessarily a natural pair.**
- **Carefully define prioritization and escalation of alerts - and avoid alert fatigue at all costs. Ensure that you know what the thresholds for your metrics should be and that you have the right alerts configured for them.**
    - If thresholds are too low, you get too much noise and possible fatigue. Too high and alerts won't go off until it's too late!
- **Make sure applications are instrumented well enough (in other words, reporting the right metrics) to be able to tell whether they're doing what they're supposed to - without debug logs!**
- **Don't put alerts on the potential side effects of issues; e.g high CPU or memory usage. Instead, monitor for the impact these things might have on your application (e.g. slow response times or unexpected restarts).**
- **When emitting logs, always use structured logging. Both humans and computers need to understand logs - structured logs can be made easily understandable by a human, but unstructured logs are hard to make a computer understand.**
- **Ensure any emitted logs are centralized and that they make sense without context. Everything you need to make sense of a log message should be in the log entry.**
- **Consider using automation around common alerts; unexpected service restarts, database connection counters, service response times, etc. This will save teams the effort of figuring it out every time a new service appears (or worse, not figuring it out and not having any monitoring).**

We've left observability until last because in some ways, it's the most complicated of the operational awareness basics. Observability is a bit of a tech buzzword, but it's also very necessary and often completely misunderstood.

Let us explain.

Observability basics

Observability is the ability to tell what your app is doing in production from monitoring the external information it produces. It includes applications sending metrics, providing useful logs, having centralized logging, tracing etc.

The most important thing to remember is that it is a state or quality of your system, and not a tool or method that you can buy. That's actually a good thing, as it is something that you can work on crafting now, rather than having to wait out trials and purchase processes.

Taking observability into account, it's clear to see how the monitoring-metrics-alerting trio comes into it - all of these aspects must be firing on all cylinders to produce the meaningful information that you're going to need for true observability.

However, if you've ever tried to decipher a structured log before, you'll know that having this monitoring and metrics is not the same as actually being able to draw meaningful conclusions from them!

Having your metrics and understanding them

As soon as you start to look into observability, you'll find dozens of tools that aim to help you achieve it - monitoring tools, logging, dashboards, AI. There's no doubt that these tools help. The tools of today produce incredible amounts of historical and real time data, and you'll likely need some extra AI or machine learning to help mold that data into something you can use.

That said, it is the human ability to leverage these tools, aggregate the information, discard superfluous alerts, centralize the logs, and extract helpful conclusions that you're really looking for. The best person for this is likely to be an experienced SRE and when you find a good one, you'll see that they're worth their weight in gold.

When all is said and done, a truly observable system allows you to work in harmony with your tools, allowing you to preempt and prevent problems, rather than just wading in when something has already gone wrong. If you reach this point with your teams and systems, you've done your business a major good turn. Take a second and congratulate yourself, because it isn't easy, but it definitely is worth it.

SECTION SUMMARY

- Once teams are operating autonomously, you need to ensure safety
- This must be as automated and natural as possible
- There are several aspects
  - Backups
  - Redundancy
  - Version control
- Techs need to be able to be flexible when it comes to solving problems
- Monitoring needs to be as smooth as possible for maximum efficiency
- Your whole approach to safety must be based on the concept of observability, which you need to research and understand now

RECOMMENDED READING

Resilience Engineering in Practice: A Guidebook —Jean Pariès, John Wreathall, Erik Hollnagel

Behind Human Error — David D. Woods, Sidney Dekker, Richard Cook, Leila Johannesen, Nadine Sarter

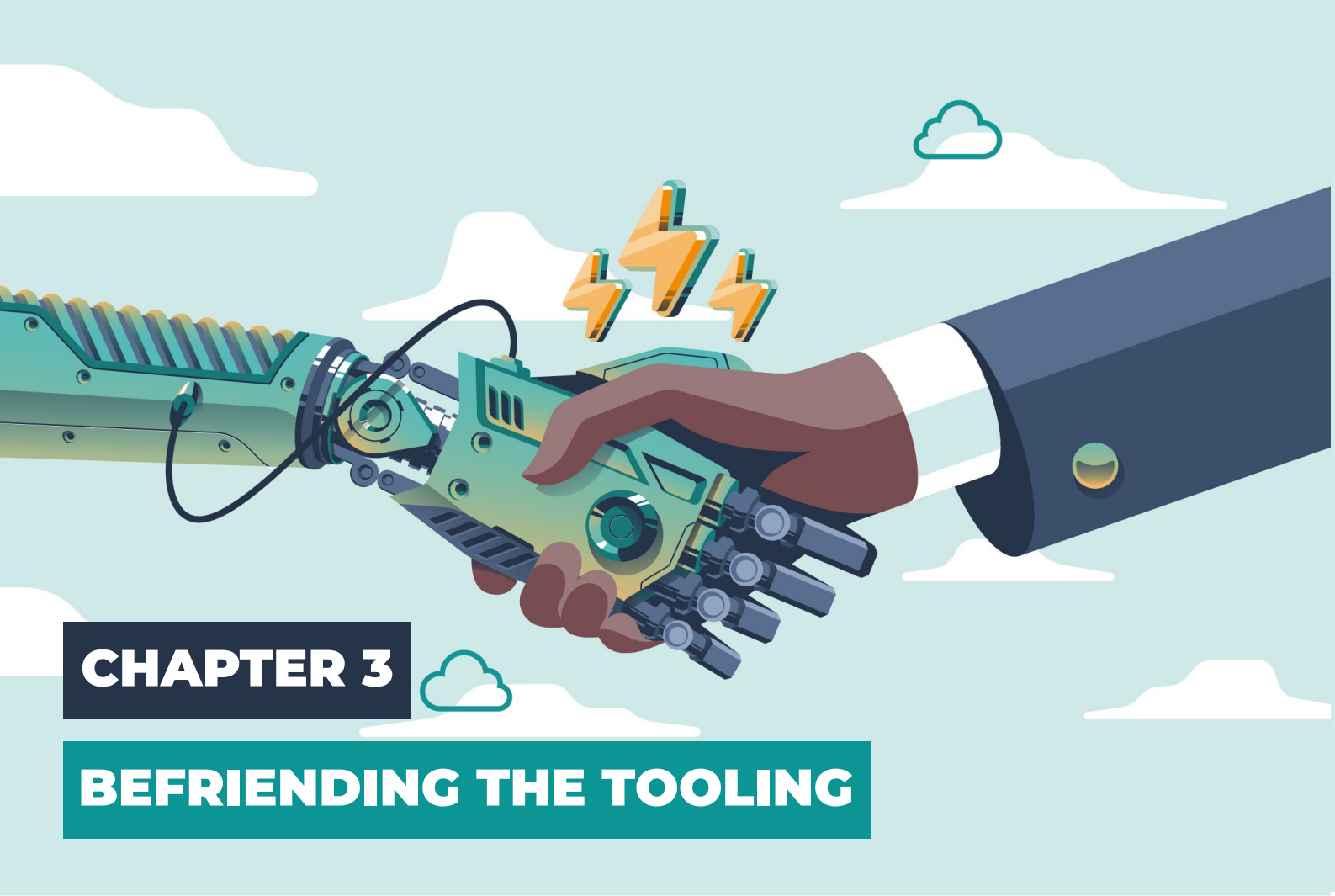
Web Operations — John Allspaw, Jesse Robbins

Database Reliability Engineering — Charity Majors, Laine Campbell

The Site Reliability Workbook — Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara and Stephen Thorne *(free to read online)*

Site Reliability Engineering — Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy *(free to read online)*

Building Secure & Reliable Systems — Heather Adkins, Betsy Beyer, Paul Blankinship, Ana Oprea, Piotr Lewandowski, Adam Stubblefield *(free to read online)*



CHAPTER 3

BEFRIENDING THE TOOLING

You're going to find that, sooner or later, you need to adopt some new tools to help you move towards a new, automated future. That's normal and to be welcomed, but with the same caution that you'd welcome stray cats to your garden - one is fine, but that can rapidly grow to something you can barely make sense of, let alone control!

The first step to keeping tooling under control is to make the adoption of the new solutions a deliberate and considered decision. No tool should be chosen spur of the moment, impulsively, or made on somebody's recommendation - however well it worked for them.

Once decisions are being taken in relative leisure, there are a few points to keep in mind when it comes to picking a candidate and, if you apply them every time, you'll have a much greater chance of adopting tools that work for both you and for your team.

Research before you try, try before you adopt

Learn the tooling before you even try it. It's really important that you spend the time you need on research before you even have a possible in sight. Take the time (and get expert help) to recognize and verbalize patterns in your use or needs. Make sure you can articulate the problem that you're hoping to solve.

Learn how the tool in question solves this problem and how it fits in with your set-up. Before you even try it, read the documentation, watch the videos, and assess the reviews. Ask things like - how does the tooling expect to solve the problem you have? What is the ideal solution to your problem? Does this tech work well with the tools we already have in place? Ask questions like this and ensure that the tool in question (or its salespeople) provides an appropriate answer.

Once you've made a decision, take every available opportunity to try it before committing. Even the best research can suggest a tool that sounded great in theory, but nightmarish in practice, through nobody's fault. If you don't ensure this before you welcome it into your tech stack, you'll end up with 100 seats for the next 10 years on a tool your team can barely use!

Make sure you can work with the tool

This is closely related to the "try before you buy" point, above, but it is worth mentioning as a standalone concern. Make sure you're willing to adapt to the way the tooling works before you choose it - a tool can tick all the boxes in terms of features, but if the way it must be used is incompatible with the way you and your team do things (a specific requirement or even a dev preference), you'll find yourself fighting an uphill battle that will often end in wasted time and money.

Ensure nobody fights the framework

It's really important that your new tool or system allows everyone to work as calmly and fluidly as possible. Your devs' tools should be an extension of their fingers and not a contraption that slows them down and demands workarounds at every step.

The best way to ensure this is to do your research well, but also watch out for:

- If you suspect techs feel the need to go outside of the tooling, stop. There's a high probability that you've missed something and you need to take the time to find out what it is before you end up with shadow IT problems.

- If somebody announces that limitations and restrictions are required to work with the tool, listen carefully but remain aware that the more limitations, the more likely you are to see problems emerge.

With regard to this point, if it becomes obvious that there's a genuine need to provide limits in how techs can interact with the tool, consider delegated access. Delegated access is about providing a path to privileged actions in a secure way without actually giving users the permissions themselves. Permission, which may or may not require approval, can often be delegated through the tool itself, so check to see how your chosen tool does in terms of role management and credentials.

A piece of advice: *use approval gates as a last resort. An approval gate is a stage in a CI pipeline that requires approval (usually manual) from a person or script before the pipeline can continue.*

Many devs find them annoying or feel that they cheapen the other approval gates that really matter. In reality, it's a balancing act. Devs, like all humans, react badly to controls they feel are unnecessary. At the same time, however, they often appreciate controls where things are complex or potentially dangerous.

The best you can do? Try to get the balance right, solicit feedback, and tread softly when it comes to locking processes down.

Tool changes ONLY

It's important that everyone agrees on tool-only changes where possible from the very start. If there are specific use cases for which your new tooling cannot be used, make sure they are clearly defined and very infrequent (think of database cases, upgrades, and root changes). If changes *do* happen outside the framework, one person should be responsible for going back and updating the tool. The vast majority of the tech team should not have the ability to do this - so keep it locked down, but ensure it happens.

Secondly, ensure that any changes that do have to happen do so consistently. This is important as if techs make changes with varied tools and in varied ways - if there are no limits - you'll never see the full potential of your new tool, as efforts to improve it are diluted across all the non-standard alternatives floating around.

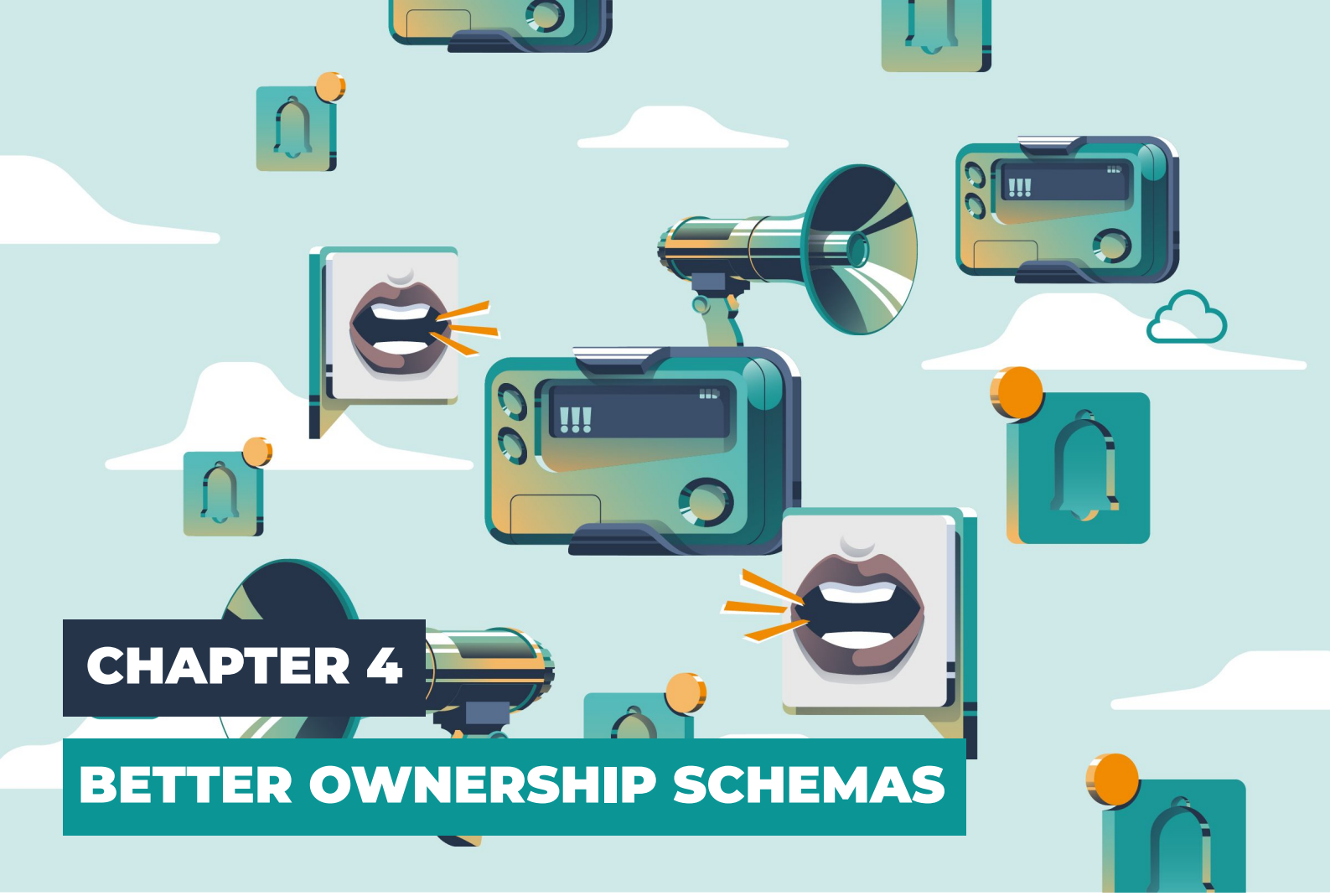
Automation first

If you're likely to do something only once, automation doesn't make sense. By the time you have to do it a second time, however, you should be thinking about how you can automate the manual steps the task contains. The third time around, you should be implementing automation such that the fourth time happens completely automatically. It's important that your new tool will support this automation, and not hinder it.

Automation-first approaches are about ensuring something can be automated rather than it actually being automated from the get-go. When you do have automation in place, you need to ensure that it's always used in preference to manual approaches. When taking on new tools, firstly make sure that they will facilitate automation, as well as supporting any automation that is already in place. New tools shouldn't end up fighting existing automation or - less dramatic but quite disappointing - simply never living up to their potential or preventing other tools from reaching theirs.

SECTION SUMMARY

- **The secret to choosing great tools is great preparation**
- **Make sure you first understand your circumstances and needs**
- **Once you understand that, the rest of the work is done BEFORE you try**
- **Bad choices lead to workarounds, resentment, and shadow IT**
- **Get into the “automation first” habit early**



CHAPTER 4

BETTER OWNERSHIP SCHEMAS

As automation allows your business to scale and grow - and there is team-wide responsibility for your product - you'll need to create an on-call framework to ensure that the quality and reliability of your product is maintained without sacrificing the health and productivity of your team.

This is something that should be assessed and quantified early, with a clear and sustainable plan for scale and change. Sure, you might have to tweak things when this change actually happens, but if you've already considered the challenges, you'll be in a much better place to do this.

Another advantage that we'd rather not think about. We live in a world where tech breaches are frequent and scary. Companies of all sizes could be hacked, but once you start to draw some attention, it becomes significantly more likely. By drawing up an incident response strategy in addition to your on-call strategy, you'll be in a much better position to handle it if the worst *does* happen.

Creating a better on-call framework

To be clear - almost all products have people who can be called when things go wrong out of office hours, but this is very different from an actual on-call policy. Without a strategic plan underlying your on-call rota, it can become stressful, unfair, and create serious tension.

At the very least, you'll need to define:

- Who is on the team?
- At what frequency is the team rotated?
- What is the on-call "onboarding" plan for new recruits?
- What is your priority system?
- How and when are alerts escalated?
- How are swaps or substitutions handled?

There's no great mystery to creating an efficient schedule once you have these things sorted out. Help your on-call team to create the schedule themselves so that it suits their particular needs and requirements. Have them define a priority system so that everyone has the same definition of an "emergency" and make sure that the escalation policies for these priorities are clear to everyone.

That includes people who might sometimes get forgotten - depending on the company, non-technical staff like customer support or managers might be needed to evaluate the impact of an alert or even to fix it.

It goes without saying that you need to make sure that roles are fairly rotated. Doing this ensures two opposite problems are avoided. On one hand, fair rotas make sure people are not on call too often. This keeps people mentally and physically healthy and ensures there are always appropriate backups in case of problems.

One the other hand, good rotations also ensure that people on call in larger teams get called out enough - if you're relatively new and only going on call once every 6 months, you'll never learn the skills you need to do the job efficiently.

Rotas also need to take account of the fact that your techs are humans, not machines. Your rota needs to be robust enough to stand up to swaps and changes, from extended periods (new baby at home, illness, stress) to just a couple of hours (unexpected pickup duties, doctor appointment). Make sure your alerting system can support these kinds of changes and overrides.

Alert fatigue

Make it your absolute priority to ensure that alert fatigue doesn't become a factor in your on-call schema - it's when automated alerts consistently and constantly alert humans to problems that do not really require human intervention.

In the worst of cases, people learn to ignore these alerts, sometimes with disastrous consequences. Most of you reading won't have human lives at stake, but in their own way, ignored alerts could cause havoc. Once you're sure that only a "true" emergency will wake your techs from their slumber, make sure the escalation policy is well defined so it doesn't pull in more resources than are really necessary - sure, it's a problem, but does it really require 6 techs to be paged on a Saturday night? Finally, ensure that these definitions of emergency and escalation are agreed upon by everyone and documented in writing.

Major incident response

Depending on where you are in your company's journey, the possibility of a major incident can seem pretty far away. Even so, it's a good idea to plan one right now. In the best-case scenario, you'll never use it, and in the worst, you'll need it tomorrow. Get ahead of it today - even if it will change in the future and might even need to be modified while in use, it will save time and panic if there is already the semblance of a plan for what to do when the worst imaginable happens.

Get ahead of it by defining an incident response strategy that lets everyone know what's expected and what to do when things go FUBAR. To do this, take some expert advice in imagining the worst possible things that could happen (technologically speaking) to your company. Then take each and every case, and work out a detailed plan for how you would go about assessing the situation, stemming the damage, mitigating your losses, and tidying up (and apologizing) when the worst is over.

A lot of work? Yes. Better than deciding on the fly when your customer credit card databases have been hacked? Also yes. When things go terribly wrong, you'll also probably want an incident response commander who has helped draw up the strategy and knows it inside out. This person will act like a drill sergeant, giving orders, allocating resources, and generally keeping an element of calm and order in a horribly stressful situation. You'll thank us later.

On-call bliss, now and in the future

Another important step you can take when it comes to on-call is to lead by example and make sure your on-call team is a stress and drama-free zone. Eliminate blame from the equation and have regular simulations (or trial runs) to at least attempt to ensure the smoothest, most stress-free system you can.

Finally, monitor and track the performance of your on-call team. Investigate metrics and get feedback from individuals. You'll see patterns that you can act on to further improve and optimize the team.

SECTION SUMMARY

- **Your on-call schedule will make or break your tech teams' growth**
- **Plan now to mitigate future stressors**
- **Involve the tech team in planning**
- **Strategize for major incidents before they happen**
- **Think about an incident response commander**
- **Monitor and assess to optimize and "future proof" the schema**

RECOMMENDED READING

The Healthy Programmer — Joe Kutner

Managing the Unmanageable — Ron Lichty, Mickey W. Mantle

Righting Software — Juval Lowy

Software Engineering at Google: Lessons Learned from Programming Over Time: — Hyrum Wright, Titus Winters, and Tom Manshreck

The No A$$hole Rule — Robert I. Sutton

Data incident response process — Google *(free to read online)*

ARTICLES MENTIONED

https://www.computerweekly.com/feature/Choosing-the-right-software-infrastructure-to-support-your-operations-mobile-needs

https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/?view=azure-devops

https://www.learnupon.com/blog/choosing-software/

https://www.atlassian.com/incident-management/on-call/improving-on-call

https://blog.newrelic.com/engineering/on-call-and-incident-response-new-relic-best-practices/

CONCLUSION

Congratulations. The fact that you are here now, reading this, means that you are interested in making changes and establishing a protocol for things that will serve you in very good stead, no matter where the future takes you.

The great thing about DevOps thinking is that it doesn't stop where DevOps doing comes in. Whether you pick a DevOps framework tomorrow or in two month's time, you can continue to foster and champion DevOps thinking in your teams, as it will benefit them regardless.

As you'll have noticed the closer you move to DevOps thinking and automation, this way of working really ushers in a new way of conceptualizing work. Your company may be further along or behind on this path to work, and you may have a harder or easier time ushering in these new ideas, especially as they relate to teamwork, collaboration, and autonomy.

That's why such a large part of a seemingly "technical" ebook is taken up with ideas that are distinctly non-technical in nature - it's because human collaboration and teamwork are the base on which all automation is built. Don't be fooled - the bright robot future of automation is 20% robot, but 80% human!

cycloid

www.cycloid.io