Being a digital leader is not just a practice; it's personal. You put your heart and soul into it. In this guide, Adriaan has packed decades of hard-earned experience in how to wrangle DX in a large organization – and make it move fast.
– Rasmus Skjoldan, Chief Marketing Officer, Magnolia

FASTER

A GUIDE TO SPEED FOR DIGITAL LEADERS

ADRIAAN BLOEM

A GIFT FROM MAGNOLIA

FASTER

A GUIDE TO SPEED FOR DIGITAL LEADERS

ADRIAAN BLOEM

MAGNOLIA

FASTER – A Guide to Speed for Digital Leaders
Adriaan Bloem & Magnolia
mgnl.io/faster

Layout: Pernille Sys Hansen
e-book versions: Torben Wilhelmsen

PREFACE

Everybody going fast in our industry – started off slow. There's no one way to accelerate and you just can't magically go from 0 to 60 in no time. When velocity does arrive, maintaining it then takes a whole different set of skills. *FASTER* is a series of personal stories, bringing you insights and expertise from the playbooks of some of the foremost DX practitioners out there. These are their hard-earned battle scars, now available to everybody wrestling with complex digital operations in an industry that only values one speed: fast.

First up is Adriaan Bloem, fresh from more than a decade in the Middle East heading one of the world's largest modern digital operations. Adriaan's guide to speed for digital leaders is a personal account of his experience, but his lessons are universal. It's rare we get to peek behind the curtain at how the most successful orgs arrive at and maintain a winning tempo. This is one such peek – we hope it's of use. Ready to pick up the pace?

– Rasmus Skjoldan, Chief Marketing Officer, Magnolia

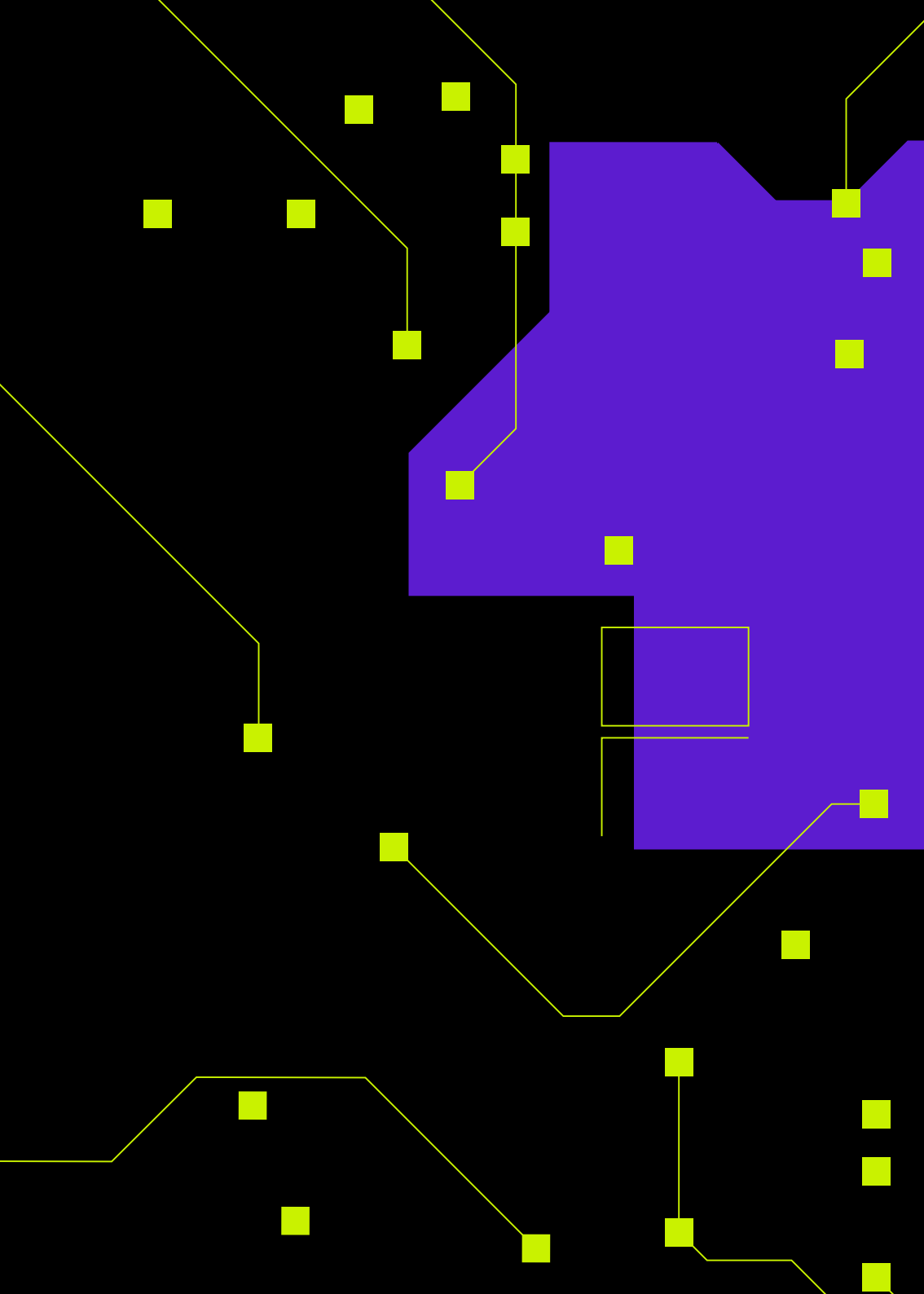
Adriaan Bloem is first and foremost a digital leader. He has worked in digital for over two and a half decades. He's followed a path from webmaster, sysadmin, developer, project manager, and industry analyst – all the way to heading one of the world's largest modern digital operations. He was the head of digital infrastructure at the largest video streaming service in the Middle East for 10 years, scaling it to millions of subscribers. He now works with organizations across the globe from his home office in Spain. Adriaan's favorite activity is swimming with penguins.

We've chosen to make this book available using a Creative Commons license (CC BY-SA 4.0). We believe that innovation comes from sharing thoughts and ideas. And given that the theme of this book is speed, it made the most sense to us to make it as widely available as possible.

So here it is – may it help you go faster!

CONTENTS

1. Introduction 8
2. Revolution 14
3. Evolution 33
4. Conclusion 56



1. INTRODUCTION

In 1989, Tim Berners Lee invented the World Wide Web, which became generally available over the course of the next two years. At that time, I was still messing around with bulletin boards, IRC, and Gopher. I didn't pay that much attention to the Web until a friend of mine showed me the Mosaic browser in 1993. It suddenly clicked. Here was something that combined three things I really liked: code, design, and writing! My code was (and is) pretty messy; my design skills never won me any awards, and I hope my writing has improved since then. But it grabbed my attention, and ever since, I've been involved in online projects in one way or another.

Of course, what started out as 'the web' immediately took off. It has since changed the way we communicate and do business. From the early amateur days in academia, it rapidly professionalized. Those original three areas of interest grew into specializations, and then those specializations further diverged. Whereas in the early nineties a 'webmaster' could (and had to) do everything themselves, there are now thousands of different areas of expertise. I'm still trying to have in-depth knowledge of each of those diverging branches, but I increasingly have to admit that the best I can do is "I know what I don't know". But it's fun trying to, so I keep learning.

We started out with 'the web', with pages edited in raw HTML. Then came the WYSIWYG editors – the first content management systems. And nowadays, we all know this isn't

simply about HTML pages anymore. Or just managing content and publishing it on a website. The information flows into many different channels, and even the simplest site will use a dozen different tools to manage it. What matters now is the digital experience – DX. That term brings the focus where it should be: the audience, users, and customers.

Managing that experience is difficult, not in the least, because it needs to look effortless on the outside. It just needs to work, like magic. And as any magician will tell you, that's really complicated. A project manager once asked me, "Adriaan, why is this so hard? I have the feeling we must be doing things the wrong way." But no, that wasn't the case; getting it right happens to be really hard. Actually, the better your understanding, the harder it gets. Once you get past the first peak of the Dunning-Kruger graph, you know how much goes into the best magic tricks.

This isn't particularly helped by the fact that the field evolves so quickly; we're constantly aiming at a moving target. There's no room for a slow and steady approach because by then, the goalposts have shifted. Which also means that beyond the questions of "what do we need to build?" and "how will we build it?", an important recurring theme is: "how can we do this faster?"

How to improve digital experience velocity

In the boardroom, the question you'll get will be, "why is this expensive re-platforming project taking so much time?" In daily operations, it will be "why is it so time-consuming to build and maintain these integrations?" or "why does it take so much time to get this campaign live?"

The more experienced your team and the more cycles of re-platforming you've gone through, the louder the question reverberates. Over the decades, I've spent a lot of time mulling this over; if technology is constantly advancing, and the people involved know what they're doing, then what keeps slowing us down? It's a question I've discussed with people over beers

at conferences, over coffee during many projects around the world, and of course lately – during video calls when we had a few minutes left to chat about the broader issues.

So rather than try to create a comprehensive, all-encompassing guide to "everything that goes into successful digital experience management," I decided to narrow down the topic. This won't be another Content Management Bible or 'polar bear book'. I'm assuming that, if you're reading this, it won't be your first rodeo. You'll have plenty of understanding of how to tackle this and don't need someone to hold your hand. So instead, it's everything I think impacts speed on digital projects; illustrated by anecdotes taken from actual projects I've worked on. It's an informal attempt at answering that question. "How can we do this faster?"

Revolution and Evolution

If you're working with a DXP to manage your digital experience, there are two very distinct phases in the lifecycle of the platform. You set it up, and then you operate it. That sounds deceivingly simple, and everyone who already has a running infrastructure will know just how much effort goes into these stages. To do that more justice, I've labeled them 'Revolution' and 'Evolution'.

First, I'll talk about the Revolution. This is where you get the buy-in and go ahead to re-platform your DX. The focus there tends to be on the RFP and technology selection, but before that, I take a step back to look at the wider organization and how it impacts the speed of the actual implementation. I'll then give you some thoughts on the planning and project structure, and then continue on to the actual requirements and platform selection.

Launching a new DXP infrastructure can be an enormous undertaking. Once it goes live, you really don't want to consider doing it again, any time soon. For that, you'll need to move from Revolution to Evolution. I'll cover this from two main angles. First, infrastructure changes, including how to add major inte-

grations, new services and tools, and change your implementation. Then, daily operations, which includes actually using the platform day to day, how technology can support that in a better way, and of course, how you can speed things up.

My hope would be that if my advice is helpful, and the stars align to actually follow it, you won't have to cycle back to another Revolution. Instead, you can successfully maintain a high-paced Evolution. If you do, let me know! I would love to get tips on how to get better at this. As I mentioned before, I don't claim to know everything – and what I know, I've learned by experience, by asking lots of stupid questions, and listening to great advice.

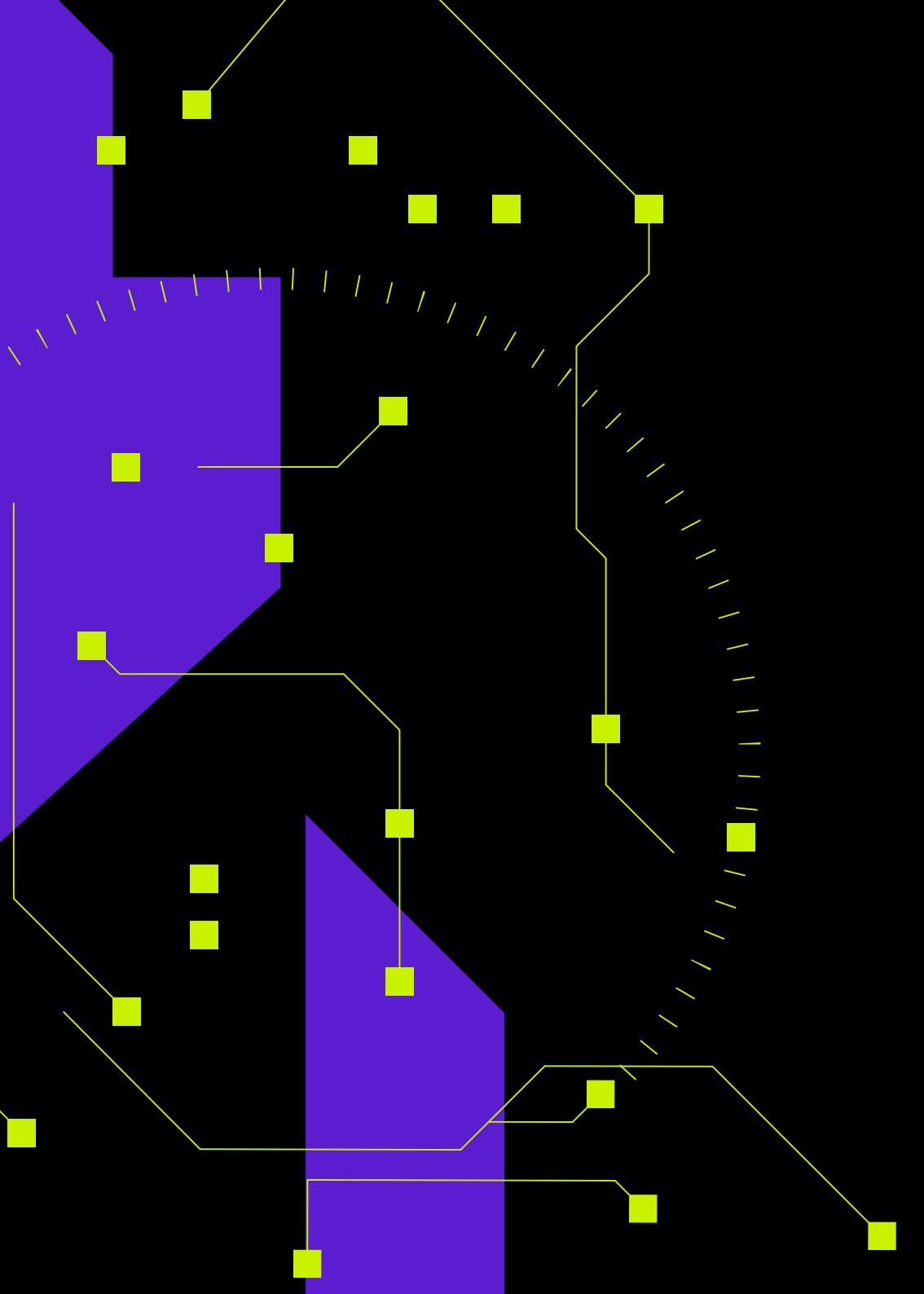
Who is this for?

As mentioned, if you're reading this, you probably have plenty of experience with this already. You've platformed and re-platformed since before 'digital experience' was even a thing. Hopefully, you'll recognize my anecdotal evidence and find it useful. Other than that, I'm not specifically addressing any level of hierarchical seniority. A friend of mine once showed me his new business cards. "Look!" he exclaimed, "I have no job title anymore! Now I've really made it."

Similarly, this is not about any specific type of organization. I've seen similar patterns in small and large enterprises, non-profit, government, NGOs, education, and large multinationals. Consequently, I'll use words like 'organization' or 'company' almost as synonyms – it really isn't about a particular legal entity or org chart; the common denominator here is digital experience.

And finally, I quite like the term 'digital experience', though I'm less enamored of the somewhat buzzwordy acronym 'DX'. But it's still useful in specific circumstances to break this down to more tangible tools and services. 'Content Management' is still an important practice, and a Web Content Management System a useful tool. Calling systems 'digital experience platform', or DXP is good when it serves to emphasize the goal (dig-

ital experience). It's less helpful if it's marketing and intended to lead you to believe it will automatically do everything for you. There's a lot more that goes into DX than just a system, platform, or suite, and you'll probably need many more than just one tool to achieve it.



2. REVOLUTION

The lifetime of a DX implementation is usually around three years. Around that mark, markets have shifted, marketing wants a refresh, and a new project will be started to redo the branding and design. This will usually encourage looking into deeper platform changes, replacing that old CMS, and adding new or better digital marketing tools. In short, fix everything that was broken and get the shiny new toys.

2.1 Organization is everything

Torturing the CIO

Before you begin, there are some hard questions to ask. As a consultant, flying in and then taking the elevator up to the 56th floor, if I have the chance, I will do what I refer to as "torturing the CIO" (borrowing a joke from Four Weddings and a Funeral). It basically means asking one question:

"Why do you even have a website?"

Of course, this could as well be the CEO, the CTO, or the CDO. And most of these people don't get to their position without being able to answer a question like that, even if they didn't expect it. Yet, it's not a very fair question because, of course, every company or organization needs a digital presence. The simplest reply would be it's simply the cost of doing business. After all,

an office also has phones, and most people aren't particularly interested in calculating the ROI of having those on the desks, either – you simply need them.

But surprisingly, the answer would be improvised, overly verbose, and pretty indirect. That's a good early indicator of a problem. If the different parts of an organization that have to cooperate on digital experience don't have a shared understanding of the core goal, we can't expect them to quickly move in the right direction. My equally unfair follow-up question is, "how does your digital experience link to your mission statement?" It is an equally unfair question because while most organizations have a mission statement, it will often be aspirational marketing.

In short, the *real* question that needs to be answered, as early, concise, and direct as possible, is "how does your digital experience impact the goal of your organization?".

Depending on the industry, this can range from blindingly obvious (an ecommerce or online gaming company lives or dies with the digital experience); to very indirect. I once asked a university Professor these questions, and he replied, "well, this institution has done very well for centuries without a website." This was true, but entirely unhelpful as guidance for a major re-platforming and digital infrastructure project.

Spend a moment to think this through and make sure it's clear to everybody involved, from the CEO or President down to the marketing intern. I'm a great advocate for having a 'digital experience mission statement', and it should be as practical as possible. If everyone is eager to start running, it's important to run in the same direction. It's equally important to understand where you're running to. I've seen too many projects that were objectively a success in their goals but achieved little more than what can be described as 'distinctly meh' in terms of results.

Measure everything

You should, of course, attach KPIs to your digital experience mission statement. This too can be blindingly obvious ('sell more ads', or 'ship more widgets'). But in many cases, you'll have

to dig a little deeper, especially if DX isn't in the core DNA of your organization (yet). However, even with the centuries-old university, where I was assured profits were not the goal, it turned out to be relatively easy to start listing out the main KPIs: 'increase student enrollment', 'improve the international ranking of research', 'reduce the use of paper', and so on. (The final list of 'core KPIs' was 7 items. Don't overdo it.)

A KPI isn't much of a KPI if you can't measure it – you'd want to have a before and after for a major platform update, and if possible, be able to express the ROI in hard numbers. However, there are 'soft' and 'hard' KPIs; and then, sometimes, 'soft' KPIs can be converted to 'hard' KPIs if you take a closer look.

Let me illustrate with an example. What often starts out as "our editors hate the current CMS" is not unusual – and probably very counterproductive – but it isn't exactly measurable. The soft KPI would be "let's poll the editors on their opinions of the current system; and then let's do that again, once they've settled into the new system." That would at least give you a 1–5 ranking on aspects of *their* digital experience in the platform, and if (and by how much) it has improved after the investment. But it would still be hard to link this to your organization's main objectives. As much as many companies value the employees' happiness and well-being, that's not what will keep the ship staying afloat. A valid point would be to say that employees are paid to put up with it – it doesn't have to be as easy to use as Facebook just to make it fun.

However, 'editor productivity' can be costly to ignore. If you time daily tasks, you can measure them again once you present the new environment. You could even calculate the actual cost and how much your investment is paying off. Suddenly, a tangential sentiment becomes a very tangible economic driver.

When the needles don't align

Inevitably, you'll run into the next hurdle: the larger your organization, the more divergent the goals of various departments will be. That's far from a digital problem (and I could fill another few pages with how DX projects often inadvertently

unearth various hidden problems that were covered up until 'disruption' came along). But it can be a major drag on the speed of what you're doing. What people, teams, and departments are judged by (and what drives promotions and bonuses) is often contradictory. A simple one would be: "IT needs to reduce cost and improve infrastructure reliability" vs. "Product wants to add many new features for the users". As straightforward as this one seems, you can already see how it would cause many conflicts in direction between just two groups. When you then extend this to more groups, and with very implicit or barely articulated drivers, trying to run a project at speed can start to feel like running through quicksand.

So, you could (and should) probably find enough measurable soft and hard KPIs for the project. You should also group them and rank them hierarchically. Don't think of this as wasting time – it will pay off once you move on. And be sure to explicitly unload some of these goals when they're not that impactful in the bigger scheme of things. If you want to have cross-departmental buy-in, make sure conflicts are resolved, or at the very least, acknowledged. I once asked the CMO of a media company, "so do you want to have more visitors on your site or more viewers for your channels?" Those can be conflicting targets, but the answer, of course, was "Yes!" That's often the reality of these projects – but at least, in this case, it was identified early on as a complex contradiction, which got the attention it required.

Time is relative

You don't have to be Einstein to know that time is relative, and as Douglas Adams put it, "time is an illusion; lunch time, doubly so." Departments and groups not only have different targets and goals than the organization at large, they also have a very different sense of what is 'fast'.

For instance, updates on an internal financial system can be glacial – it's much more important that the consistency of core reporting is safeguarded, than supporting the interface on the iPhone 42 or Android Liquorice. Meanwhile, users of a large

B2C platform will have no such salaried patience, and will go onto Twitter with pitchforks and torches if they can't complete a transaction after an update of their device.

I remember working on a subscription service and being on the phone with my DevOps lead. We were laughing, "well, obviously, if a query takes longer than 30, the call center would explode!" After the call, a colleague told me, "I'm sorry, I couldn't help but overhear – that's interesting. We've been working on improving the query response times on the Oracle ERP. They're also taking about 30 seconds. So we're working on the same problem!" I didn't have the heart to tell him we had been discussing 30 *milliseconds,* not seconds. Painting the canvas on the app and filling it with carrousels and posters takes many API calls, and if it would take 30 seconds to show anything to a paying subscriber, they would throw their phone against the wall in frustration. (In fact, it had been an actual user complaint, and the user wanted to be reimbursed for the broken device.) For an employee to wait half a minute to see how many holidays they have left may be annoying, but if you'd employ the 'hard' KPI mentioned earlier, it would be difficult to make the business case to spend money to improve it. There was no real ROI on massively scaling the ERP clusters.

That's not just an amusing anecdote. Various parts of your organization move at very different speeds, and usually, for very good reasons. This can be a serious problem for the time to market of a digital experience platform, and you'll slow down when:

- You adjust to the speed of the slowest common denominator.
- You optimistically fit projects to deadlines instead of time required.
- You let the most agile and eager teams set the pace, and 'dependencies' don't catch up.

In practice, it's not just different speeds; it's entirely different timelines, as well. Finance will be eager to close the books at the end of the fiscal year; Sales may want to get signatures before Christmas; and your iOS developers are trying to get the

new app version out before the Apple keynote in September. Ignoring these differences will turn harmony into discord, so you need to be realistic in your planning.

Chop up the elephant

A friend of mine joked that the biggest problem for mammoth hunters was fitting them in their fridge after the hunt. And digital experience can be the wooliest of mammoths to fit into a project. It's challenging to set your DX mission, reconcile conflicting KPIs, and then bridge multiple timelines. The answer is to chop up the elephant and use multiple refrigerators. Cut the mammoth down to size.

1. **Divide digital experience re-platforming into multiple projects,** and assign them specific KPIs.
2. **Allow the projects to run on their own timelines,** and remove as many interdependencies as possible.

For example, a CRM team needed to urgently address customer complaints in a more efficient way. The team was strong (and transparent) in their reporting, and reasonably requested the implementation of a CRM tool be given priority. This team was given its own budget, project group, and timelines. As a result, it was implemented in several months – instead of having to wait for the completion of the second phase of the overall product roadmap the quarter after.

The risk here, of course, is that you lose coherence and oversight. This would be a good moment for me to state that I really love mammoths and don't like to see them cut to pieces, either. But in reality, the extra overhead of coordination and planning pays off in speed – as long as you're well aware that faster is often more complicated. While I've been referring to a 'project', it's better to have a Program Manager oversee many projects than have a Project Manager drown in the permafrost.

Note how I coyly skipped over how difficult 'removing interdependencies' can be in practice – especially on the technology infrastructure – but I'll revisit that point later.

Who takes the lead?

That addresses the project or program, but what about the overall lead on digital experience?

In Marketing, the lines between 'traditional' and 'digital' marketing have blurred to the point that we're now talking about 'experience'. Many marketers will actually say, "isn't *all* marketing digital?" And of course, in most verticals, there are very large areas outside of digital (which leads to the inevitable overlap between digital experience to customer experience, and multichannel to omnichannel). But nobody can ignore digital anymore.

Likewise, as an organization, you could either be the 'old school' established incumbent or the disrupting startup. But either way, digital will be a large part of it, and everyone has their eyes on the big platforms that are swallowing entire industries. You want to be Uber, not the taxi company; you want to be Netflix, not Channel 9; you look at Tesla, not at burning compressed dinosaur fuel. So in a way, isn't a large part of *all* business now digital?

Reality hasn't quite caught up with the vision here. To summarize:

1. Digital experience often doesn't have a strong enough link to the overall organizational goals.
2. Different departments pursue different goals, slowing down change.
3. Different parts of your organization live at different speeds and timelines.

The buzzword 'disruption' has been overused, and the realization is sinking in that this is actually a *change management* problem because disruption is very much outside of the comfort zone for your organization. Switching technologies and platforms is disruptive, but not necessarily in a good way. You can roll out swathes of new technology, but this won't 'automagically' change your processes. Instead, you risk disrupting the

existing processes without first optimizing and changing them. Only then should you roll out the tech to support that.

So as you embark on the odyssey of re-platforming, who takes the lead? This will often depend on company culture, historic fence posts, and how the project is perceived or who's sponsoring it. In most cases, one department will be seen as the logical flagbearer. For instance:

- IT, because a lot of technology is involved
- A PMO, because of the complications of the project
- Business departments, because they need it to generate revenue
- Marketing, because it's about external communication

This is by no means an exhaustive list. I've even seen Procurement take the lead in some cases because while IT would posit that it was a technology project, Procurement would argue it was a lot of *new* technology. And who else would run the RFP?

You'll probably be familiar with the DIKW pyramid or one of its variants. One short version would be that you need data (D), but raw data needs interpretation in order to become information (I). In contrast, information is useless if it isn't received and used as knowledge (K). I once worked with a semi-government organization that, perhaps inspired by the pyramid, had a Department of Data, a Department of Information, and a Department of Knowledge. Of course, the Department of Communication and the IT Department considered themselves to be the W in the pyramid (Wisdom). It took several months to determine who had the biggest stake in the project and would therefore lead the way.

Determining the lead can be hugely time-consuming and risks putting too much emphasis on a single set of KPIs, with one timeline – which may not be the one size that fits all.

So do you need a CDO? This was a big debate fought out in blog posts and social media, but it seems to be simmering out. Having a 'Chief Digital Officer' would be fairly redundant. As mentioned before, *everybody* in the organization should constantly be considering digital. And CDO will now often mean

'Chief Data Officer' (though the rationale for that kind of CDO and the interdepartmental battles that follow will often be strikingly similar). In reality, it may be an imperfect solution to a problem that's still very real. I've seen many savvy CDOs navigate their companies to build bridges between departments and point everybody in the right direction. Having an effective CDO could be the only way to get things moving.

At the very least, the question should be, can we effectively reconcile the different concerns? You can temporarily rely on external consultants to guide you – but it may prove hard to internalize that way. If you don't have a CDO, you still need someone on the C Level as the project sponsor. Given how digital experience tends to cut through the organization, without a sufficiently senior sponsor, it will be doomed from the start.

2.2 Planning

In order to start planning a project, you'll have to choose a methodology. However, in many cases, that question never comes up because the methodology is determined by either one of these two factors:

1. You use whatever your organization is currently used to.
2. You use something different than what was used before, because projects kept underdelivering or failing.

It's easy to find any number of research papers pointing out that IT projects have abysmal success rates, and the larger and more complex they are, the worse the results. Generally, 10–25% of projects will fail altogether, and no more than 25–40% will be an outright success. For Digital Experience re-platforming, the numbers tend to be worse – given the factors described in the previous chapter.

This is why organizations have often made changes to their project organization over time. 'Waterfall' is associated with long projects, in lengthy (waterfall) phases, top-heavy on up-front planning. 'Agile', by contrast, is perceived to spend much

less time on planning and instead allow for constant course adjustments as you go along. As a result, many have abandoned waterfall methodologies and moved to agile – and then within agile, have switched from Scrum, to Kanban, to Lean. However, a skeptic would say that in practice, the common denominator is the persistent failures. A cynic would say the main reason agile success rates are higher, is because waterfall is better at defining the difference between failure and success.

As the Crowded House song goes, "Wherever you go, you always take the weather with you." If you don't address the underlying (organizational) issues I described before, you can change methods all you want – but it's still applying a method to the madness. Most organizations have been burned by the inflexibility of mammoth waterfall projects in the past, but please remember this:

Agile is not an excuse for not knowing what you want.

Long ago, I ran a consultancy with a business partner, and he was taking on a project in the Nordics. We'd often be called in to try to 'fix' large projects that were already well underway to failure. (A colleague once called me "Mr. Wolf", after the Pulp Fiction Character who comes in to clean up after a particularly bloody accident. I took it as a compliment.) This project was no different as the company had embarked on a complete re-platforming and redesign, building everything from the ground up. They had expected to complete this in six months. So I asked the obvious question – how come they haven't launched anything yet, after four years? "Because they're using agile."

Digital experience is a moving target. It's constantly and rapidly evolving. This is why we consistently see two things:

- **Waterfall projects** miss the target because by the time they finish, **the target has moved**, and the project isn't flexible enough to adjust.
- **Agile projects miss** the target because **the target hasn't been properly defined**, and constant course adjustments move the project wildly off course.

Agile works extremely well for the 'evolutionary' stage of your infrastructure and platforms (which I will get into in the next chapter), where constant adjustment allows you to align with business needs. But it can be a disaster for the underpinning infrastructure re-platforming.

This can be obvious if you look at the classic project management triangle. Scope, cost, and time are at odds with each other. Agile favors cost and time; scope can be expendable (and changeable). But for a foundation, scope and quality are rarely where you'd want to cut corners.

You could say the Leaning Tower of Pisa was built in agile. The above-ground stories have been beautifully crafted out of marble. As the tower started leaning during construction, higher floors started compensating for the angle (which is why the building is actually somewhat curved). But that couldn't fix it anymore. It had already gone wrong in the foundation they started with.

Had the foundation been built with waterfall, the tower wouldn't have been one of the famous metaphors for failure. But waterfall favors scope over cost and time, so building the understructure would quite likely have gone over deadline and budget. Given the problems with the further construction it caused (and the fact it took nearly 200 years to complete anyway), that would still have been preferable in the grander scheme of things.

This is a long-winded way to illustrate that methodologies are tools. You need to choose the right one for the job and the stage where you are at. Don't discard 'old hat' methods to jump into hipster project management trends. But also, don't simply keep doing what you were doing because you are used to it (and have the diploma on your wall). Each has its own advantages and disadvantages.

For instance, PRINCE2 is a popular waterfall method in the UK and Western Europe. It offers a lot of structure and templates to manage (complex) projects. At the end of each phase, it recognizes a 'go/no go' moment – and it's incredibly useful to have an explicit decision on whether to proceed or cut your losses. (The project can still be considered a 'success' if it gets

to a 'no go' halfway through, as long as it followed the correct process.) However, within the phases of the waterfall, it can be very hard (nearly impossible) to change direction, and stages can easily be 3 months long.

As a completely different example, Kanban famously has its origin in the "Toyota Production System". It's a great way to prioritize features and maintain a constant output. A Kanban board is also a great visualization of where you are and what's going wrong. But it's highly serialized and better suited to a going flow than to get things started. It's very useful in running DevOps and goes well with CI/CD. Toyota optimized the building of millions of cars this way, but nobody ever built an offshore drilling platform using Kanban.

Scrum, on the other hand, refuses to define itself as a methodology. Instead, it focuses on roles ('accountabilities') and 'events' (such as the 'daily standup' it popularized). This is a great way to improve the autonomy of teams and allows them to make constant, on-the-fly changes as a project progresses. However, it's entirely unsuited to deadlines. That critical functionality you need to launch with? Oh, that's still on the backlog – the Product Owner deprioritized it.

I could go on about several other methodologies. The key thing is to understand strengths and weaknesses. People can be quite religious in their debates about this and completely stuck in carefully going through the motions of their chosen path. I once worked in a PRINCE2 project with three people, where somehow the project manager saw it fit to divide us up into four different teams (each team had the same three people, in different roles – I was lead on two of them). I've seen a daily scrum that couldn't start because someone was still sitting down (instead of standing up), and which lasted exactly 15 minutes (in the middle of a sentence) because "that's what the book says". (This is where the Scrum Master becomes the Scrum Police.)

The two biggest mistaken beliefs here are:

1. Waterfall is old school and evil.
2. Agile will magically fix everything. If it doesn't, it's because you're using the wrong agile.

So what happens in practice? Most organizations will still have project managers because projects need to be delivered within scope, on deadline, and within budget. Most project managers will then have to carefully manage stakeholders to convey why you can't have all three. Managing against a deadline is best done in waterfall. However, most of the teams involved (and definitely any tech or development team) will want to work in agile (often, in scrum, because that's how they've set up their teams). That's a paradox that needs careful management rather than denial.

Be eclectic in your methodologies, and don't try to fit square pegs into round holes. The re-platforming stage is where many project teams come up with new terms for what works for them. I kind of like 'Scrumterfall', for instance (because it sounds like a James Bond movie, and I want to be Q), but I've also come across 'Fallban' and 'Lean Waterfall'. This mix and match approach works especially well if the team members are thoroughly interested and well-versed in very different methodologies. Simply trying to copy or mimic what other organizations do rarely works. Remember that Spotify, famously, doesn't actually follow the 'Spotify Method' themselves (and they'll be happy to tell you there actually really isn't such a thing).

By contrast, I remember discussing the original Agile Manifesto over coffee with a Product Owner in Finland. "If you go back to the actual core ideas there – couldn't you do agile in waterfall?" he mused. And of course, you could, if you "value individuals and interactions over processes and tools." Project management method mash-ups should be a thing. But make sure it's based on experience and knowledge.

If you try to fit everything into waterfall, *or* if you try to fit everything into agile, you risk being slow, missing deadlines, and not delivering what's expected and needed. It's perfectly fine to have a Kanban board in your Scrum, and you'll probably

have to deliver in stages to meet the external deadlines. Most of the tech giants don't force one methodology on teams, and for good reason.

2.3 Platform 'don'ts'

Volumes can be (and have been) written about how to go about selecting a new DX platform, but there are some general caveats – things that can easily drag out the promised "3–6 months" to do the new implementation to 1 or 2 years (and sometimes more). (Incidentally, this is why as mentioned before, the actual replacement cycle is 3 years, but dragging out projects often stretches that to 5 or more.) These are some of the most important ones you will want to avoid, and I'll give some extreme (but real) examples to illustrate. Be honest with yourself when you read those examples, though. Are you sure you don't recognize some of it?

Specifying mostly 'everything that is wrong' with your current infrastructure

This is often driven by 'blaming the system' thinking. It will risk turning your RFP process into a negative trip down memory lane, with requirements driven by 'fixing what is broken'. In one extreme case, I saw an RFP with over 900 questions. Most of them to be answered by yes or no, all of them derived from wish lists compiled by various teams, but almost none of them have any bearing on what the organization would actually want to achieve. Needless to say, this is not the most productive way to progress.

Over-specifying the requirements

This is time-consuming in itself – but also will quite likely lead you to:

a. Rinse and repeat the previous cycle (sometimes leaving execs wondering why they bought into this expensive and resource-intensive project in the first place).

b. Redo the requirements several times, slowly, in multiple stages, zeroing in on the goals as ideas are added or rejected, and goal posts move.

As mentioned before, you can easily spend years this way without actually getting anywhere. Long ago, I once worked on a major implementation, and to my surprise, it took over 10 years before it was replaced (instead of the more usual 3–5). Rather than being proud of my successful groundwork, it made me suspicious. When I inquired why it had taken so long, the answers were 'rapid prototyping' and 'working in agile'. Remember that rapid prototyping can be a slow disaster given the complications of the underlying technical infrastructure; agile shouldn't be an excuse for not knowing what you want to do.

Boiling the ocean

If you're experienced and savvy about what you want in a new digital experience platform, a big risk is wanting, well, everything. And with the momentum of a re-platforming project, preferably, wanting everything at the same time. But remember the woolly mammoth. Implementing a digital experience 'suite' may sound like it will give you all you need in one neat package. But in practice, they result in projects that are too big to succeed, and the individual components of a suite are often much weaker than 'best of breed' players in each area. So even if you do manage a successful implementation, functionality is likely to be below par in key areas. The other way round can create a similar problem. If you 'tie together' many specialist solutions, you can end up with an incoherent jumble of 'loose integrations' (and a lot of overhead on your front-end apps). Make sure you figure out how to properly integrate this on the backend as well. Your employees can be quietly screaming at their screens if publishing a new campaign requires hopping around the interfaces of multiple different tools.

Under-specifying the requirements

If your RFP is a two-pager, you're likely to omit some of the most important make-or-break essentials. I once saw a compa-

ny in the Middle East end up with hosting in France, an integrator in India, and a US-based vendor with a system in Java – while all of their local in-house resources were firmly entrenched in PHP. While any one of these factors could have been mitigated if there would have been a strong case for it, in reality, these weren't conscious decisions. They were caused by a casual RFP driven by financials and timelines with a mostly aspirational brief. The project was abandoned a year and $250k later.

Starting from a green field with blue sky thinking

Refusing to be limited by technology is great for the initial brainstorming on where to go, but once you get started, the sky isn't the limit. Remember that most systems are quite particular in the way they work well. But this is often because they codify years and years of experience with how others have usually set things up and run them. Make use of that experience, and don't try to use a hammer on a screw. I've seen companies spend hundreds of thousands on 'customization' trying to do everything in a slightly different way than anyone else before. It's reminiscent of the Life of Brian; "look, you're all individuals!" – but are you sure you want to be *that* particular?

Roll your own

Most of all – and this can't be stressed enough – the only reason to build your own tools is to discover why you shouldn't be building your own tools. It's easy to spend millions of dollars that way, and unless you're honestly going for that moonshot of creating the next Facebook or Amazon, be very critical of the potential ROI of "let's build it ourselves" or "let's have an outsourcing company build this for us".

2.4 Platform 'dos'

It's easy to point out the pitfalls, but this leaves the obvious follow-up question. Great, so what *should* we do? As touched upon before, a great deal will depend on the actual goals to be achieved and aligning on KPIs. But in an ideal case scenario,

not constrained by time or resources, there's a sequence that generally delivers good results. Of course, in reality, there are always deadlines and budgets, so it's fine to compress these stages by combining several into one or timeboxing the efforts. But to weed out potential problems early and reach the required results as quickly as possible, it's crucial not to skip any of them – or, more realistically, too many of them. In many cases, some of these steps are mandatory because of legal compliance or company policy. Turn that into an advantage, rather than lamenting the time lost on bureaucracy.

A classic selection process will look something like this:

- Create a long list of suppliers
- Send out a Request for Information (RFI)
- Create a shortlist of suppliers
- Send out a Request for Proposal
- Select the winner

What you send to the potential suppliers is, ideally, as open-ended as you can get away with. This takes a lot of confidence as you won't have the easily demonstrated 'objectivity' of scoring against a long list of requirements, and you'll have to invest much more time in investigating how integrators and vendors will make a platform work for you. But you'll get the best responses if you have an open conversation about what you need, and your requests should be based on scenarios that don't dictate an exact way of supporting them. Think of these as 'user stories', but more elaborate and much more of a story. A supplier should be able to demonstrate how they would go through these tasks in their systems.

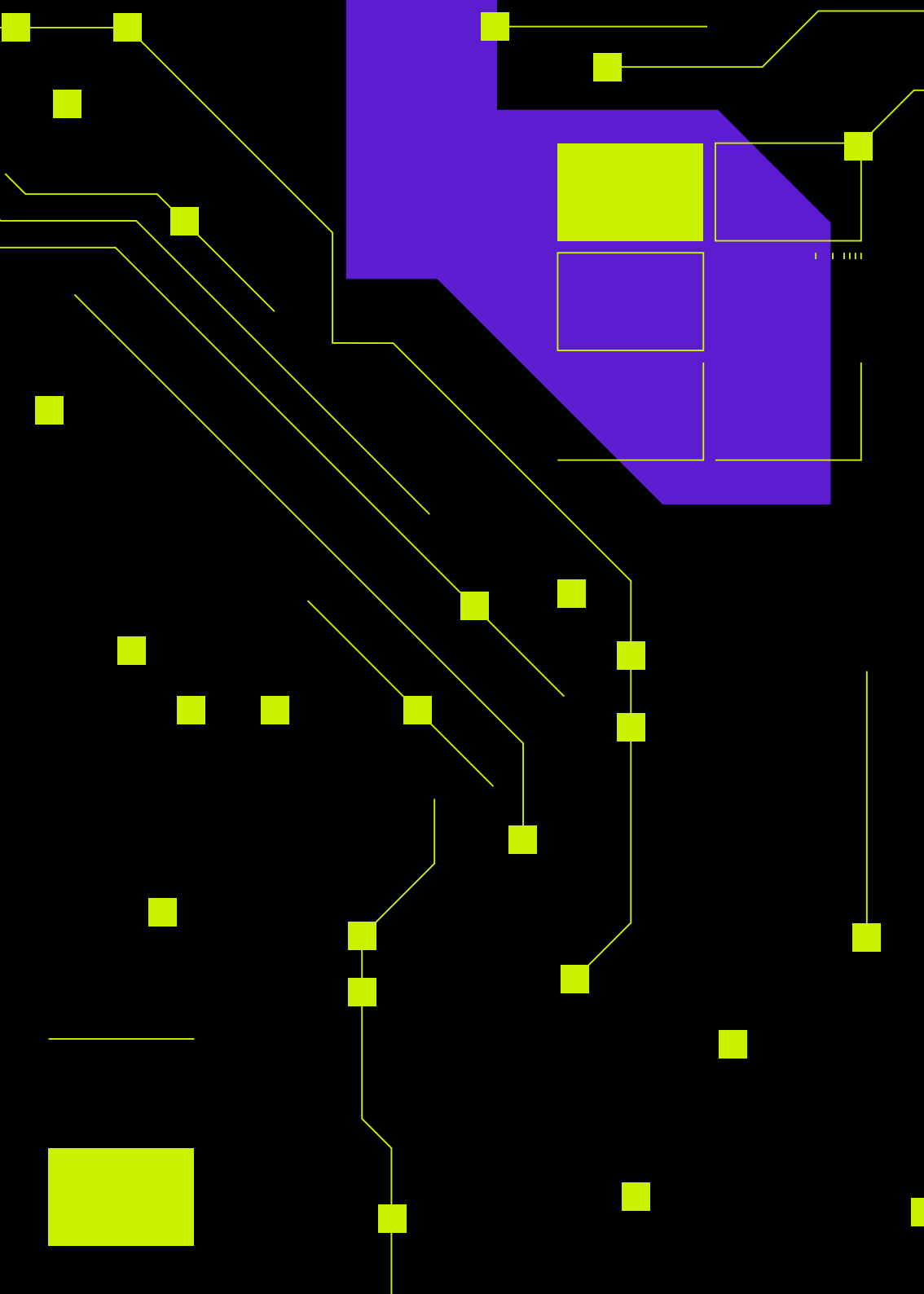
In addition to the standard RFI/RFP, I'd also encourage you to add two stages:

- **A proof of concept bake-off** before deciding, with two or three competitors: Allocate a time window to have the suppliers start the actual work of setting up your platform. To level the playing field, you should be prepared to pay them for the effort if necessary. Make sure what you set

as requirements for the PoC is actually doable in a short period of time, but bear in mind this is not your project kick-off. You also want to evaluate what it's like to work with the supplier, so you'll have an idea of how the actual project will go.

- **A discovery phase** after deciding, but before you start, and if possible, before you finalize pricing: You may think you've already set the exact requirements, and want to hold the supplier to deadlines and cost while delivering what you want. In reality, at this stage, if your environment has any amount of complexity, suppliers won't have any idea what you need. This leads to over-promising and under-delivering, and frustration on both sides.

Adding stages will feel like you're losing time, not speeding up. But in practice, especially in highly complicated DX platforming projects, a few weeks spent on these are likely to save you months of delays.



3. EVOLUTION

Once you have your shiny new platform up and running, you'll have to start changing it. I've had to remind multiple companies that launching a new site and back-end infrastructure is not like publishing a book. You don't send it off to the printers, and it's not going to sit in a library for centuries looking the way it did when you finished it. DX channels constantly evolve. In fact, this will probably start even *before* your initial (re)platforming project finishes, when new requirements start coming in even before you launch.

I tend to see 'evolution' as ongoing modifications in two distinct groups:

1. **Infrastructure changes:** When you need to add new channels, new tools, integrate a back-end system, and so on; and require (major) development.
2. **Daily operations:** The daily and weekly operation of the platform, such as adding a landing page or microsite, changing an article, adding or modifying a product, using the interfaces of your system, and without having to code.

If you master these successfully, you shouldn't need another 'Revolution' again – you can keep the entire platform up to date with constant updates, small and large.

3.1 What's major and what's minor?

If you consider 're-platforming' the most major (and disruptive) of changes, 'infrastructure changes' are next, while 'daily operations' are simply the going concern of operating the DX platform. Yet it's common to see these mixed up, and it's usually a bad sign. It's very easy to let infrastructure changes escalate into re-platforming; or to let inefficiencies in daily operations go unaddressed, like death by a thousand cuts, instead of addressing them in infrastructural changes.

For instance, if creating a new landing page requires DevOps and heavy tech involvement, what should be 'minor' is suddenly 'major'. I worked with one company where creating a new microsite would take about 2 weeks, including adding new database tables and deploying code. Since new microsites would be needed about 40 times a year, against tight deadlines, the teams were constantly fighting a backlog (as well as stressing out and generally not enjoying their jobs all that much). This was exacerbated by needing both tech, editorial, and design involvement, most of it in a very serial process, and of course, with each team having their own sense of speed and priorities.

If you'd only need this a few times per year, there's a good argument not to invest too heavily in tooling to automate a process like this. Depending on your platform, it could be very costly to customize the tools to hand control over the process back to editorial power users. However, if it's as frequent as 40 times a year – it should be operational, not infrastructural. And you should invest in the infrastructure to support the operation.

Of course, infrastructure changes then always risk turning into re-platforming projects. Adding an integration (such as a booking engine, CRM, inventory management) should be part of the *evolution* of your infrastructure. It shouldn't really require a wholesale replacement. If it does, you should go back to the previous chapter – and make sure a new infrastructure and organizational setup allows for more flexibility.

In the case of the 40 microsites, it was clear this should have been an operational process. To fix that, it turned into various infrastructure projects that never quite solved the problem.

It then became a re-platforming project which took a year to complete (costing over a million dollars). It's worth it to be strict about the levels of change in analyzing this – and it should be clear to all teams what the difference is. It can avoid years of frustration and hidden losses, but the opposite is also true – it can stop you from going to a 'big bang' every time a serious problem is identified.

There are two areas where special mention is deserved concerning the boundary between 're-platforming' and 'infrastructure change', changing design, and upgrading the platform.

Revamps, refreshes, and redesigns

'Changing design' sounds innocent enough, but it rarely means you simply update the stylesheet and change the images. In most cases, templates will have to be rewritten (which is front-end development work and quickly gets into back-end code). It often requires major customization (or re-doing the old customizations from scratch). And if the entire UX is refreshed, it will almost certainly turn into a re-platforming project. Sometimes that's merited, but it's important to go in with eyes wide open. The impact is often severely underestimated.

I've often heard it being referred to as a 'redesign' or the even more innocent sounding 'revamp'. To management, that sounds like a quick win, showing progress. In practice, you should think of it in terms of interior design. Are you painting the walls, replacing the carpet, and moving the furniture around a bit? That's a revamp. But it will often also require tearing down some walls, moving doors and windows. And before you know it, you're basically building an entirely new house from scratch. Don't let that happen accidentally.

As a side note, always bear in mind that if there's one thing your users absolutely hate, it's a major overhaul of the UX of their favorite services and sites. Even if it was obviously broken before, they learned to live with it. And even if the new UX is objectively (and measurably) much better, it will take time for users to get used to using it. It's why Craigslist still looks like it was built in 1995. It's also why eBay faced a furious backlash

when they switched their background from yellow to white, forcing them to roll back the change of color. (They then, cleverly, slightly changed the tint week by week until the yellow had become white – and nobody was upset.)

Platform upgrades

Another severely underestimated 'infrastructure change' is upgrading your back-end platform. As much as platform vendors and integrators will try to sell you on an 'upgrade', a new major version of a platform (say, from 4.7 to 5.0) can mean moving to an almost entirely new system. 'Upgrade scripts' and migration tools rarely make this a pain-free exercise. In fact, an upgrade will often be as torturous as re-platforming, especially if templates and integrations have to be redone.

It's not uncommon for back-end platforms to be behind at least one or two major versions because of this, which becomes an increasing problem given interdependencies with operating systems, databases, and other components and services. The outdated platform will often rely on outdated support acts and won't play nice with the newer, safer, and updated libraries and frameworks. This, of course, is a ticking time bomb.

So, make absolutely sure what the impact of the upgrade will be. Consult with other customers about how they fared and what their approach was. This will give you a good idea of if it was, in fact, just a simple patch, or whether it was more akin to a complete ground-up rebuild, complete with a migration project.

If the upgrade is such a major overhaul, you should take the opportunity to revisit how the platform stacks up against your requirements. But always make sure the incumbent is part of your evaluation. You may have started to hit various walls with it, but you'll be keenly aware of its limitations (and a new version might fix them). You should also be careful not to throw away the years invested in experience with your experience platform, not just the tool itself but also the vendor, the integrator, and the ecosystem.

Finally, while that all may sound daunting, don't ignore the reality that this will happen on a fairly regular basis – check

the history of your platform for major breaking changes and their frequency. Postponing a disruptive upgrade because you can't spare the resources or time will just mean the decision will be taken for you when you least want to deal with it. Once your old platform version finally grinds to a halt, fails to keep scaling with traffic, or worse, gets hacked, it'll be too late to do a considered and thoughtful evaluation of the next steps.

3.2 Infrastructure changes

Beyond those caveats, there is a good part of running the infrastructure of your DX platform that should be evolutionary. This is the more 'technical' part of keeping your digital experience up to speed. It will greatly vary how integral a part of your organization this technical maintenance and evolution is, depending on what business you're in and how crucial digital experience is seen as being to your company.

In fairness, most organizations are not, and never will be, tech-only companies, so you need to be careful what your examples are. For example, this is what most of the digital people will routinely use as examples, and it will start sounding like this is what everyone should aspire to:

- **Big tech:** Companies such as Amazon and Google, that are dominated by tech, to the point that they've productized their own platforms as cloud platforms that are serving most of the internet.
- **Major online services:** Such as Netflix or Uber, that are also quite tech-centric, and will to some extent, be building their own infrastructure but often still rely on infrastructure vendors to run on.
- **Online startups:** These will often begin with just a good idea but will quickly have to scale their tech operation once they gain traction, and they'll often be very vocal about their way of doing things.

But if you're honest, does your organization fall into any of the above three categories? Most likely it won't, and that's not a bad thing. Where it goes wrong is if you try to emulate what you read about these examples, and then try to apply it 1:1 in your own setting.

Where this can be seen clearly is in the grand aspirations that shine through in job descriptions. Large multinationals are asking for an 'entrepreneurial spirit' in digital, while in reality, actual entrepreneurs are rarely cut out for influencing decision-making in a complicated corporate environment. I've seen many strong traditional businesses ambitiously trying to poach "preferably with experience at Google or Uber" (or the equivalent in their industry) to lead their efforts, ignoring that many of the people from those backgrounds are often not going to be very effective if you take away their product and tech teams, and the custom tools they built.

Having spent a fair bit of time in media, let me take that as an example of a particular vertical. The yardstick there, of course, is Netflix, which means I've often repeated:

- **You're not Netflix, and that's fine.** There's no need to re-invent the wheel on core components they had to build themselves – which you can now simply buy, as a commodity, from a vendor.
- **Don't copy the way Netflix looks; copy the way Netflix thinks.** This is somewhat contradictory to what I've written above, but of course, there are many good ideas you can replicate from a company like it. Just be very selective in what has real-world application in a completely different setting.

For any industry, you'll have to be very careful which examples you'd want to emulate, and be very selective of what you want to incorporate.

The reason I bring all of this up in a section on 'infrastructure changes', of course, is also something I've often heard in video streaming. "Sure, we can do what Netflix does. If we can also have the thousands of engineers they have working on this, please?"

The size and responsibilities of your tech teams

I've worked across many different industries, budgets, and ambition levels. But even in the smallest projects, or in organizations that were most removed from being a tech company(even when all implementation or even daily technical operation was outsourced to integrators or professional services), there's still a minimum requirement for a tech team. And that absolute minimum is 2 people, because even if one engineer could do the job, you need a hot standby when they have a burnout. And tech people need holidays, too.

It's crucial to have at least that bare minimum of tech expertise in your own organization, because without it, you'll be forced to blindly trust a third party. And as capable, solid, and reliable a third party partner can be – and as much as they care about you as a customer – they'll never, almost by definition, care as much about you as they will about their own company. Or, as I've often put it, if you have no idea how a car works, you'd better hope you're lucky in the choice of a mechanic. This is why many taxi companies, even though they're not in the business of building or maintaining cars, will still have their own mechanics.

As the importance of digital experience grows, you'll find you have to expand rapidly beyond that absolute minimum. For instance, by now, it's not unusual for a company of 2,000–10,000 people in an entirely non-tech industry to employ about 50–100 people in tech. And by tech, I don't mean traditional enterprise IT, but people actually working on the digital experience. Without a serious team, it will be hard to evolve your digital experience quickly enough to keep up with your competitors, especially in an age where increasingly, these won't be your traditional competitors since disruption tends to come out of the left field.

But here, also beware that you're not just "not Netflix" or "not a tech-only company". Your company probably also isn't a software company. There is a strong tendency of in-house teams to build against requirements and deadlines, with roadmaps that don't cater to the reality of maintaining software. Often,

there is no time to go back and refactor, or to take services and optimize them. Even if there are all the fittings of CI/CD, good practices, and a strong DevOps culture, that's simply not really in the goals of the overall company.

In reality, even software companies struggle with this where it concerns their DX – the doctors often make the worst patients.

Build vs buy

A crucial factor in how much tech involvement you will have to insource is whether you choose to build or buy. This is a debate that has been raging for well over a decade now, and it's often presented as a black and white choice. In reality, of course, it's not even fifty shades of gray, but more of a rainbow. In fact, we should probably retire the question altogether now. Outside of big tech, nobody builds *everything* themselves. For all of the innovation Tesla has brought to cars, they're still using round wheels – they haven't reinvented those.

The reality of operating a modern DX environment is that you buy selectively and build lots of integrations. You'll need enormous scale to even begin considering 'building' core components. But tying together various best-of-breed tools will still involve a lot of building. And companies offering you a tantalizing DX 'suite' will, paradoxically, probably require you to build even more than if you'd sort your own Lego bricks. There are roughly three options:

- **Buy a DX suite that does everything,** but in practice will probably be an unexpectedly loosely coupled set of tools (quite often cobbled together from acquisitions). Either that or a gargantuan effort by a single vendor that still falls considerably short of even the most generic dedicated solutions. For instance, the built-in analytics will probably not be nearly as good as Google Analytics, and the email marketing will probably not come close to something as simple as Mailchimp.
- **Compile your own suite from best-of-breed solutions.** Depending on how complex your DX environment is, you'll

probably already have several back-end systems for dedicated purposes anyway. You can selectively buy additional components to extend the capabilities, for instance, adding SalesForce for CRM. However, your DX will start to be scattered across multiple silos, and teams will be driven to insanity trying to operate multiple back-end systems in order to do something as simple as launching a new campaign. The proliferation of tools will also start to increasingly impact your experience. It's not unusual for an app to have to communicate with several dozen APIs from systems ranging from a CMS, to identity management, to analytics, and social SDKs. This is why many companies end up building 'middle layers' to unify the communication from all of those systems and interface with the apps.

- **Choose one platform to be the pivot for all others.** This type of 'composite DX platform' means you can bring your own tools – buy selectively, and get best-of-breed capabilities. But the integrations will often be easier to achieve (and often there will be pre-existing integrations to work with), greatly reducing the time to implement them. It also means you can focus more effort on creating a coherent back-office environment where your internal users can operate.

I have never particularly believed in 'suites' that can do everything. I love my Swiss Army Knife, and it's great when you go camping; but if I were to build a house, I'd rather have dedicated tools. The main exception here is when you use a suite to kickstart your re-platforming. However, if that's the case, you'll need to quickly scale your tech organization to cater to buy, replace, and integrate the parts that are sub-par. At home, I really only use my Swiss Army Knife for the bottle opener, but the first week after moving into a new place, I'll use it for everything while I try to figure out where I left my screwdrivers and can opener.

As mentioned, though, the big problem with running all discrete components is that not one system is leading – with the issues described. Building the 'integrations' to tie it all together can turn into a nightmare of conflicting interdependencies.

'Middle layers' are also where the optimism of microservice architectures dies on the field. They become classic monoliths and a maintenance nightmare.

Choosing one platform to be leading and serve as the integration hook for all others is the ideal situation, but it's also a bit of a utopia, as most platforms aren't all that well suited to build against. "It has various APIs" is one way that salespeople will sometimes obfuscate the reality of what can be more of a black box than an open architecture. This is something to seriously investigate and, if possible, try before you buy. If anything, documentation for closed source software is notoriously spotty, and even large and popular open source projects struggle to keep up in that regard. And even if your platform is well defined, and architected for extension, have a hard look at how this works out in the interfaces. Something may very well be 'integrated', but that will be meaningless if it ends up like a runaway train without the driver being able to keep it on track.

Inhouse or outsourced

I worked for a great COO who had a very simple rule. "You only hire people if you intend to employ them for at least three years." While I'm a great advocate for in-house tech expertise and building out your own team to at least lead the development in the right direction, you definitely can't scale that team fast enough for re-platforming (and unless you keep doing that every year, it would make no sense). But you'll also find that the going concern of infrastructural change will have high-charged peaks against deadlines that are hard to deal with or require expertise that is hard to hire and expensive to keep idling. In other words, you'll need partners to keep evolving your DX infrastructure. There are three main alternatives to tackle this:

- Work with Professional Services
- Work with an Integrator
- Work with an outsourcing company

Working with a vendor's PS often seems like a great idea – they'll likely have the most expertise in their system, and if anything breaks, can actually get access to the core development teams that built it in the first place. However, on larger projects, they can often be stretched thin; and you may end up with a lot of boilerplate implementation code that was never actually meant to leave the sales demo lab, or hasn't been properly evaluated in a while. Even worse, if you have to get two systems to play nice, dueling PS teams from two or more teams can create a blame-shifting nightmare. On one project, the project manager told me it reminded him of the movie Gladiator, with the vendors battling it out in the arena. "And we don't actually have an Emperor who can give a thumbs up or down, either." In practice, professional services are often most useful when strategically applied to very specialist parts of a project (or the parts where a system is particularly resistant to pleasant cooperation).

For re-platforming projects, an experienced third-party integrator will often be a better mix of both dedicated to your goals and independent enough of the software they're implementing. But, here, be ready to read between the lines when tough choices have to be made or the integrator is pushing for some particular solution that nobody in your team seems to like. There can be internal conflicts in resourcing the different projects an integrator is running. Finally, never rely on an integrator to maintain the in-depth knowledge of your particular project. Employees of integrators are as upwardly mobile as the best of us, and if you have to adapt that integration built by that large integrator a few years later, you're likely to find that the original developers have moved on and nobody knows how it works anymore. The knowledge will have to be transferred, and a hand-over is not just a set of documents, but should ideally involve some intensive co-development to properly anchor it in your own teams.

Outsourced development can be even more difficult in that respect. This can be very hard to manage, especially if it's only a handful of resources – because then, the onus of managing them will fall on your own teams. With larger outsourced teams,

that will be easier – especially if your outsourcing partner also has project managers, architects, and product owners. Ideally, these will work closely with their counterparts in your company, so there's constant communication on every level. 'Throwing requirements over the wall' is another accident waiting to happen. This is particularly true of offshore development (and even near-shore). If the teams don't know each other and don't regularly communicate, outsourcing is doomed to fail. I've been on many standup meetings covering multiple continents. And while they can be difficult (with teams trying to understand each other's accents and cultural differences), it was always worth the effort.

Remember, time is relative

As mentioned before, different parts of your organization live at different speeds and timelines. This means that one department will come up with a big, disruptive request for an integration, while another is desperately trying to make the deadline for a redesign. Even when a company is fully convinced they've embraced agile, this will usually be a serial process. You can have your new integration – once we're done with the current roadmap items. Superficially, that makes complete sense, as evolution of a DX platform will often already feel like you're trying to modify a car while hurtling down the highway at 200 kph. But as a fan of fast German cars, I can tell you from experience that cruise control stops working at that speed. You can't afford to rigidly hold on to your defined Scrumterfall phases if you want to be faster.

One of the biggest tricks to learn and master is therefore to not only use *multiple lanes*, but allow the cars to move at *different speeds* and *different times*.

As an example, I once worked on a video streaming service where the entire streaming infrastructure was being overhauled and moved from one cloud vendor to another in order to improve the streaming experience; not a time where you'd want your tech team to be distracted by other requests. However, simultaneously, the entire ad tech stack had to be ripped out

and replaced, or the streams couldn't be monetized anymore. And coincidentally, the CRM team had an urgent request to integrate a CRM tool, since user complaints needed to be dealt with more quickly and efficiently – also directly impacting the overall experience.

The traditional way to deal with this would have been to first finish the video infrastructure, then do the ad stack, and then start on the CRM integration. But since all three were almost equally crucial to the service, that wasn't really an option. So instead, they ran in parallel, starting and ending in different but overlapping months. The video infrastructure was handled entirely in-house; the ad stack was tackled by an integration team working with external vendors; and the CRM integration was done with the help of an integrator.

I can't say this was pulled off without lots of noise, panic, and some chaos, but in the end, all three were finished within the same quarter. It was well worth it because otherwise, it would have been delivered over the course of a year. Lesson learned: There was no real point trying to avoid the complications of running major infrastructure projects in parallel. It was impossible to avoid. So, instead, if you want to be fast, you need to embrace it and get better at it. And instead of a swimlane diagram, use the Autobahn.

3.3 Operational changes

What I call 'operational changes' is, in reality, what your internal teams will spend most of their time with. You could simply call it 'operations', but I think it's important to distinguish it from the tech/infrastructure evolution. At the same time, this is where you'll most often feel the pain if things are slow. If your organization, as in the example I used before, creates lots of campaign microsites or landing pages, doing that quickly is worth a major investment. If you have to create lots of product variants, this should be easy to do. If you have many articles, your platform shouldn't be the bottleneck in editing and publishing them.

As mentioned, I don't necessarily think operating a DX platform needs to be 'fun'. It's one of the things I keep hearing about 'millennials' and how they're 'digital natives' that won't accept the bad design and complications of an old-school, enterprise-like dinosaur of an interface. They'll want it to be as easy to use as the apps they use at home, such as Facebook, Instagram, or TikTok.

Comparing work to leisure that way doesn't make much sense (and it's also rather condescending to millennials). If it were true, we should probably overhaul the entire office, as well. Get rid of the old-fashioned desks and office chairs, and replace them with comfy couches. Have everybody work on bean bags while simultaneously playing games on big video screens and eating pizza. In fact, a lot of companies have tried exactly that – but it hasn't always improved productivity or even morale. During the lockdowns of the pandemic, we've all had to learn to work from home, and by now it's becoming a problem to get people back into the cushy offices that tried to be a better version of the home environment.

The reality is that operating a DX environment is complicated and hard. It's not at all as simple as posting your own story on social media. DX lives in multiple channels, often in multiple languages, and addresses multiple audiences. The experience is not just two or three dimensional; it will usually have at least five and probably more dimensions to create variants for. And you need to take the experts who run the operational changes across this for you very seriously.

So operational change is a complicated beast, run by professionals you pay to get it right. But while there's no need to coddle, it's still exactly the area where a DX platform will need to prove its value. Back-end interfaces don't have to be hip and trendy, and they often don't benefit from the same iterative investment in polish that large B2C platforms offer. But it's also where you're most likely to lose or gain speed in evolving your experience. Optimize your back-end environment, but be ruthlessly rational about where you want to put the investment.

Let me give you one example. I worked with an organization where publishing an article would take at least two weeks. We

counted the steps required, and we came to an astonishing 37 clicks. That was just the bit supported by the system, though – the actual process also involved emailing Word drafts to Legal, then to translation agencies, and then the editor copy/pasting it into the back-end interface. Then, within the system, it would pass at least three people before it could get online. In fact, what was even more astonishing, is it could even be done in two weeks.

Before the industry embraced the term 'DX platform', we would be talking about 'Content Management Systems'. And every once in a while, someone would attempt to define what a CMS actually is – not an easy task, since even defining 'content' is quite difficult. The best I've ever managed to do was this: "A Content Management System is a system to *support* managing content."

This is what often causes most of the pain in operations, forgetting the platform needs to support the actual process. Because of that, it doesn't magically transform the process for you, and it won't automatically do everything for you. Worse, quite often, it doesn't so much support as actively impede. Instead of feeling the speed of running carried by the wind, the platform puts up an obstacle course. But how do you fix that?

Analyzing the process

Before you begin to rebuild or change your operational interfaces, you'll need to analyze the existing process. The initial question is usually, "but why is the process so slow?" I mentioned the arduous article publishing process. I've seen similar problems elsewhere.

In one environment, getting an asset online would often take two days – even though they were supposed to be live the same day at 8pm. Nobody could quite put the finger on the delay – and the teams responsible were working very, very hard to get faster at it. After going over every single step, it turned out there were two main inefficiencies that were surprisingly time-consuming: hardware bottlenecks and a serial process for editing the asset in different language variants. The hardware

bottlenecks were relatively easy to fix, though the fix was costly; just throw more bandwidth and processing power at the problem. The editorial process, of course, couldn't really be fixed in software or platform updates unless the actual process was changed first. And in that case, something that was originally seen as out of scope and rather unrelated turned out to be a major factor; the process simply needed more resources.

In another company, publishing campaigns would take months. In fairness, the campaigns needed to be rolled out in dozens of languages across about a hundred countries. Translation and compliance (both legal and cultural) were, of course, time-consuming. This was, again, a serialized process (from language to language), and the main hurdle turned out to be the translation agency (which often didn't have the capacity to do multiple translations in parallel). Once the process for that was fixed, major infrastructure changes allowed the platform to actually support the new way of working.

It's essential to draw out actual flow charts of the process, both outside of the system and within. To gain speed, you'll want to find things such as:

- Serial processes that can be parallelized
- Undocumented processes that live outside of the platform
- Inefficiencies in the operation of your platform

It's useful to actually calculate the time spent on overheads like this to then be able to put a number on 'hidden cost' (wasted time). It makes for a great 'hard KPI' that can be measured, and it makes it possible to do an ROI calculation on the investment to improve the infrastructure and process. This could be as simple as multiplying cost per resource by the time spent. You then have a very immediate cost reduction to aim for. It's often an easier argument to make than "it'll improve our competitiveness" or other imprecise goals.

As you do the analysis, be sure to be critical as you can't always count on informal accounts of the steps. Many times when I was explained the process by a manager, the actual, real-life process turned out to be wildly divergent from what

was designed or even perceived to be the case. You have to sit down with the people doing it, using a stopwatch, and observe what is keeping them. You'll see them switching between interfaces, to email and Word and back, and there's a lot of phone calls that have taken the place of system notifications that don't quite work.

Also, bear in mind that the people within the process will often not know how to solve it – or will feel powerless to do anything about either the process or the system that is supposed to support them through it. I was once sitting in a bar, late at night after a conference, with a fellow geek as we were talking about fixing processes, and I asked him, "Do you mean – asking the users how to fix it?" He exclaimed, "Oh lord no! They don't know. That's not their job." Which I thought was fair enough – but it does mean you need someone with a broader view and mandate in a position to do that job.

Work flows - or does it?

Workflows deserve special mention. In theory, workflows in a DX platform are there to enforce processes and keep work, well, flowing. In practice, they seriously slow that flow down to a trickle, and are generally a terrible idea. You should never have anything more than a 'four eyes' workflow – meaning, one person creates, another checks and publishes. There are only two exceptions to this:

- Translations
- Compliance

Translations were mentioned in one of my examples. To take it a bit more broadly, anything that requires multiple variants created by humans, and especially if it needs to be worked on by external partners, can benefit from good automated workflows directing the content back and forth through that process. Just make sure the workflows are 'non-blocking'. You don't want the entire publishing flow to halt because one of 27 languages isn't ready yet.

Compliance is the only good reason for approval workflows. If you work in pharma, medical, insurance, or banking, you'll want to have everything checked and double-checked before it goes out, because it may be a considerable liability if you don't. It really needs an official sign-off before it can be made public.

In practice, most approval workflows are there because of a lack of trust. I once had a conversation with an editorial manager who was designing a complex set of approval and rejection flows, and told him to remove all of it as it would turn publishing into crawling. "But what if one of my editors makes a mistake?" he said. "Well, you correct it and publish it again." "But what if one of them maliciously slanders the company?" he asked. "Well, you correct it and publish it again. And then you fire him." And, of course, that should be something a good manager should see coming long before it goes off the rails like that.

At the same time, a good platform will, of course, help in managing editors, as well. It's useful to have good audit trails of who did what. It's also important to see how well content does. And it should be part of the daily practice to review and discuss how to improve. There are various excellent tools you can add to your platform to help with that, depending on the level of sophistication you want to take this to.

But it also reminds of a broader truth that applies in most stages and aspects of DX:

You can't be fast without trust.

Word, Excel, and Outlook are still the main DX tools

Another inconvenient truth for your operations is that most of it will probably be completely outside of your DX platform. The Office tools are a good example. There's a reason that 'copy/pasting from Word' is still the bane of many platforms (it's notoriously hard to properly clean the Office metadata that's pasted without accidentally getting rid of all the formatting). Most articles are still written in Word, most pricing lists and schedules are still done in Excel, and most images will still come from Photoshop.

This is perfectly understandable, and not just because everybody is so used to it. As I touched upon before, a lot of processes will actually take place outside of your system. "Quickly emailing someone a Word doc" is the most common way of getting feedback and approvals. But there are various other reasons, as well. I remember an implementation that a company was particularly proud of. They had managed to be nearly 'feature complete' in the editorial interface, based on extensive user interviews. Yet everyone was still writing in Word and then transferring into the system. As it turned out, many of the editors were in the field, working through spotty internet connections. The one feature the platform didn't have was autosave. Every time an editor would lose their connection, they would lose their entire article and have to start over. You need to find all of the shortcomings of your platform if you want to move the process from Office to the cloud.

Of course, you could very well question why you'd want to enforce that in the first place. And in many cases, it's actually perfectly fine, and you shouldn't waste enormous amounts of time trying to get people to adopt an online editor over their trusted offline tools. (Remember that the Achilles heel of cloud productivity suites is still connectivity, so using Office 365 will actually be worse than using it on a PC.) However, in many cases, it will hold you back in the age-old page thinking of yesterday. This can be a huge obstacle in modern digital experience, because it will make it very hard to re-use content across channels and repurpose it for different audiences. By mixing form and content, you'll be stuck with the original formatting, sometimes to the point that you can't do a redesign without then having to do a migration to clean everything up.

Modern online editors allow you to create structured content that's properly split into sensible components that can be identified and used by machines, not just 'a bunch of pages'. Having a more atomic description of products and articles not only allows you to safeguard the value of already created content in the evolution of your DX. It also means you can publish it to other channels you may not have thought of yet. And if

anything, it means you can mark up HTML with microformats and get Google to notice.

To summarize, Word has no place in DX; but it will probably be around for a while.

Previews: The disaster you won't see coming

If you have staff working on campaigns, landing pages, articles, products, and so on, you'll want them to be able to imagine what this will actually look like to the consumer of the content. This has always been a problem in a digital environment, because you don't have precise control over the output (the way you'd have in print). I once made relentless fun of a designer whose daily routine included recalibrating his very expensive wide-gamma screen to be more accurate in color reproduction. "You do realize most people will see this on an old, yellowed CRT with bad phosphors, right?" Those bulbous cathode-ray screens have gone the way of the dodo by now, but one thing remains, you have no control over the resolution, brightness, color settings, screen sizes. A lot of very subtle design is totally lost on many terrible displays. And with apps, new problems are introduced every few months. I've lost a month on a product release because the app logo turned out to be hidden by the then-new iPhone XS' 'interesting' notch on the top of the screen. You can't control the end users' devices, so you need to bake in flexibility and fallbacks.

Modern digital experience, of course, is a much bigger problem than that. Not only can't you control the display, or the display size and resolution, the content will escape your sites and apps. It will live on in multiple channels, including social media, where each service has its own bright ideas on how to improve the experience. When Instagram introduced stories, they weren't square anymore but portrait. And when TikTok started supporting landscape orientation, and then changed it a few times, it caught many publishers by surprise.

The classic 'preview' of most DX platforms is woefully inadequate to account for all those points of view. At best, it gives an indication of how, for instance, a web page might look on a

mobile device. But will it still look the same when it's displayed on Facebook's embedded browser on a phone? And if you're looking for the real horror stories, go talk to people who have to design email newsletters. HTML support in most email clients seems to have frozen in the world of Internet Explorer 6.

This means two things. First of all, don't overdo it; you can't possibly accurately preview on every single platform and device, so allow for some disappointment. But secondly, and more importantly, make sure your staff is well aware and builds up the experience to understand how digital experience traverses the channels.

This becomes even more important in the 'five-dimensional' world of modern DX. Because apart from channel and device, there's also a temporal flux. One illustration of this is what I've dubbed the 'Christmas problem' in ecommerce. One day everybody is looking for trees and ornaments, and suddenly the next day, nobody is interested. Similarly, there are one-time events, special promotions, licensing, and a host of other reasons why content has a window. How can you possibly preview every single version at every single point in time?

Again, this will mean relying on the expertise of the people working the backends, and making sure they have the tools to quickly and efficiently set those parameters. In terms of previews, consider them hints and reminders. Rather than trying to reproduce everything, give many good hints and focus less on the verisimilitude. It's more useful to see the biggest variations instantly than being able to investigate each in detail.

Avoid the multi-screen hopscotch

As has come up in other sections, one thing that is quite probably sucking up time and sucking the life out of your speed of execution is if your staff has to jump from screen to screen. To launch a product or campaign, they may need to perform tasks in a multitude of systems and screens. Find images in a DAM, write articles and descriptions, add these into a CMS or product database, create screens, banners, and sliders, add

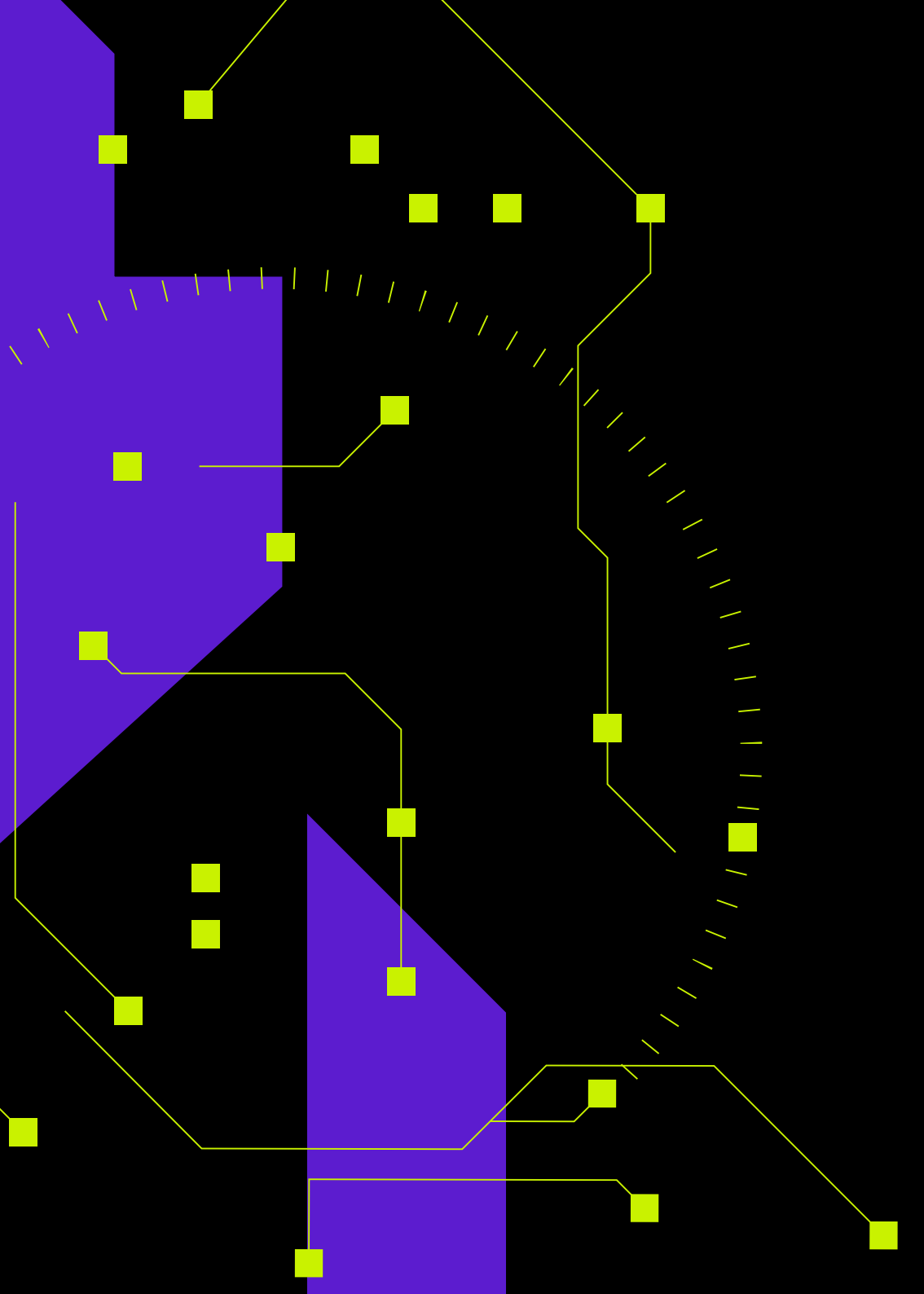
sections and links, and so forth. Better still, try to schedule all of these in different backends on the same timeline.

This introduces room for error as it's hard to maintain consistent data across all tools. It will also confuse and distract your staff. And it will be much slower than anyone involved would want it to be.

Make sure operational change lives in as few as possible interfaces, and try to consolidate as many as possible. If you have a modular DX platform, it can serve as the central hub to direct and access the other systems and tools. Switching between windows, copy/pasting, saving, and uploading drain the life out of the teams. Hopscotch is a fun game, but if the stakes are real, you'd want to go in a straight line.

3.4 Evolution as the end to Revolutions

I've mentioned several times that if you can keep evolving at a steady pace, you may be able to avoid needing another 'Revolution'. If you can keep changing your platform bit by bit, asynchronously, potentially, you won't have to do a wholesale re-platforming ever again. Of course, there will be times where that's still unavoidable – directions change, businesses merge and split, and sometimes you have to redo your entire digital experience from scratch. But even then, I'd encourage you to do it as fluidly as possible. Of course, you want to disrupt your market. But you don't want to disrupt your business.



4. CONCLUSION

There are two things I promised you in the introduction: a description of the Revolution and the Evolution of DX. More importantly, I've attempted to find the factors that slow these down; and tried to explain how you can speed them up. Maybe it can even help you avoid yet another Revolution if you get the Evolution right.

Hopefully, you've found this enjoyable to read, recognizable along the way, and it would be fantastic if you're able to use it yourself.

I mentioned the Dunning-Kruger effect in the beginning without actually explaining it. I've seen it posted and re-posted as a graph so many times I didn't want to bore you with the summary. On the Y-axis, there's confidence; on the X-axis, there's actual competence. At the left side of the graph is the peak of 'high confidence' coupled to 'low competence'; people who don't know what they don't know.

More interestingly, though, is what follows after that initial peak of over-confidence; a deep valley where competence is actually increasing, yet confidence is only slowly recovering. This is where I'd put myself, constantly learning because I'm very aware that there is so much more to understand.

If you're climbing that same mountain of digital experience as I am, I'm sure you've disagreed with many things I've written.

And that's great! Please find me on Twitter or LinkedIn, or hopefully sometime soon in real life again, and let me know. I love debating these subjects, and I'll happily concede I'm wrong (at least occasionally).