White paper

The art of designing a software factory: **Shaping your organization for success**

Mastering the Microsoft stack

mstack

The art of designing a software factory: Shaping your organization for success

At its core, software is simple: it essentially boils down to a string of ones and zeros. Software development, on the other hand, is less straightforward – or, at least, that's certainly the experience of many organizations. With the pressure on to harness the potential of ever-growing mountains of data – and to deliver solutions that will live up to the expectations of today's digital-first society – it's no wonder that companies often find themselves struggling to design and build the software they dream of.

If you suspect your business is making software development harder than it needs to be, it may be time to take a step back and (re)evaluate your software development practices. In many cases, the solution is to build an organization that can efficiently produce the right outcome at the right time. Or, in other words, to optimize your very own software factory.

Let's take a tour of the factory floor.

What is a software factory?

Just like any real-world goods factory, our example software factory consists of inputs, processes, and outputs. Here, we'll focus on the process side of things – the inner workings of the factory.

	Goods factory	Software factory
Inputs	E.g., raw materials, ready-made components, energy	E.g., functional requirements, technical requirements, constraints, existing modules
Processes	E.g., melting, molding, machining, labeling, additive manufacturing, testing	E.g., designing, building, testing, deploying, running, maintaining, monitoring
Outputs	Final product (e.g., a smartphone, toy, or car)	Consumer- or business-facing software (e.g., an app, API, or platform)

A note on product requirements

Although this white paper doesn't go into detail about inputs (or outputs), it's worth noting that product requirements constitute what is arguably the most important kind of input for a software factory. Requirements capture the needs of the business and translate them into goals, in areas ranging from security and architecture to legal and compliance. Often, they need to be refined before they can be used effectively and efficiently in our software factory processes.

It's important to keep in mind that we're talking about a modern, 2023-standard factory. These days, automation should handle the repetitive tasks, leaving your business's talented developers free to tackle the more creative, intelligent parts of the process. After all, that's where our strengths as humans lie. Of course, the automation taking care of the repetitive tasks will have been built (and must be maintained) by software development teams: your factory's expert engineers.

Recognizing the need for change

Now that we have a better idea of what our software factory looks like, we can address the biggest issue it's likely to face: operational efficiency (or, rather, a lack thereof).

To reach the perfect state of optimal efficiency, the factory would need the most effective production process in place, transforming inputs into outputs that precisely meet every business requirement and are delivered exactly when they're needed. If that sounds too good to be true, it probably is – after all, there are always trade-offs to be made and external factors (cost, time-to-market, scale, business priorities, etc.) to consider. Nevertheless, even if we can't achieve 100% efficiency in our factory, we can still aim high if we embrace these challenges and ensure our processes are flexible enough to adapt to a changing business environment.

In other words, just like a goods factory, a successful software factory needs continuous maintenance – and the occasional redesign – to make sure its processes and outputs meet the needs of the business and its customers, respectively. For many organizations, though, this is easier said than done. In our experience, even practitioners of agile methodologies often neglect to carry out the optimizations they need to keep their software development factory running at optimum efficiency.

Further reading

For more insights into agile methodologies,

see our white paper on organizational agility. →

It's worth noting that in many cases this neglect has nothing to do with any inability to bring things up to scratch: more often, it's simply that the company hasn't noticed that it needs to. So, let's pause to ask: how can you tell?

Sniffing out inefficiencies

Software engineers will be familiar with the concept of 'code smells' – clues that a codebase might be harboring some quality issues. The more code smells, such as duplicated code, you find, the bigger the red flag and the bigger the risk of a critical problem.

Likewise, we need to sniff out (potential) trouble in our software factory and its processes, so we can resolve the underlying issues and inefficiencies. In an agile organization, for example, the following might be indicators that something is amiss:

- **Development teams have adopted agile while the rest of the company hasn't.**
- **Development teams have empty in-trays or overflowing backlogs.**
- **Functional development is on hold while teams focus on internal platform issues or technical debt.**
- **Inter-team dependencies are holding up tasks or projects.**
- **Priorities aren't clear and/or aren't aligned with the business's goals.**

These signs can be hard to spot when you're right in the middle of them. That's why bringing in an external party to assess the situation with a fresh pair of eyes can be a valuable exercise. After all, over time, the engineers in our example software factory grow more and more accustomed to its ways of working. In fact, employees saying "that's just how we do things around here" is another smell to stay alert to.

Need a new perspective?

We'd be happy to cast our expert eye over your software factory. Contact us here. →

Digging to the root of the problem

When it comes to operational inefficiencies – just as with any health condition – recognizing the symptoms is one thing, but treating the underlying issue is what matters. Unfortunately, many software factory managers find themselves too busy firefighting day-to-day problems (the effects) to properly tackle their root causes – or they're simply unwilling to take the factory offline while they do.

This means that the important maintenance tasks or upgrades needed to keep the factory working at peak efficiency often go either unrecognized or ignored. Either way, they remain unaddressed – meaning our factory is missing out on valuable opportunities to improve the quantity, quality, reliability, and scalability of its outputs.

Instead, factory maintenance should be a priority, and not just for a single proactive engineer or team, but for the entire organization. Methodologies like SAFe, specifically its Inspect & Adapt phase, and Scrum, with its 'retrospective' step, encourage software factories to take a step back from putting out fires to treat the cause of the initial spark.

So, what could be causing operational inefficiency in a software factory?

Cause #1: Conway's Law

More than 50 years ago, computer scientist Melvin Conway stated that organizations design software systems that mirror the communication patterns within the organization. For instance, when a company has a discrete team for database administration, the software landscape is more likely to lean toward centralized databases; when different teams have their own database experts on board, software systems tend toward decentralized databases.

Applying this principle to our software factory, we can see that not only should our system architecture support our business requirements, but the design of our factory should mimic the system architecture we're looking to achieve. That is, we need to consider the structure of our desired outputs and organize our processes accordingly – otherwise known as the reverse Conway maneuver.

Plenty of companies, however, fail to take this into account when designing their software development processes; more often, factories are built to tackle large projects that ultimately don't deliver (or even end up being discarded when priorities shift). And only a handful carry out periodic reorganization based on the products they want to build – because the reality is that keeping up with Conway is an ongoing process, requiring an organization to continuously adapt based on regular inspections of its software factory.

Of course, this kind of change can be hard – which brings us to our next root cause.

Cause #2:
Resistance to change

Whether it's in our nature or something we learn over time, many people don't welcome change with open arms. In a business environment, in particular, people perceive reorganization as (at best) an inconvenience and (at worst) a threat. Given that many such changes are brought about by factors like a disappointing result or a merger, it's easy to see why. This is where building the right company culture becomes important. Organizations need to nurture adaptability in their employees, helping them see change in a more positive light: as a step toward their goals rather than as a response to a perceived failure.

After all, there's much to be gained. In line with Conway's observation, adjusting your organization to support the outputs you want can be key to unlocking success. Several business architecture frameworks explicitly recognize this: take TOGAF, for instance, which mandates skill management to ensure an organization has the capabilities it needs to produce the desired outputs – that is, to ensure our example software factory is well equipped to produce our required app or platform. The Inspect & Adapt phase in SAFe, meanwhile, promotes reorganization as a recurring activity (admittedly, often on a team level rather than at a departmental or company-wide scale).

How to optimize your software development

Understanding the need for change is a key first step, but there's still the issue of how to go about it, especially for non-software companies that nonetheless rely heavily on software or their software departments (every company's a software company nowadays, right?). In today's finance industry, for example, many businesses are wrestling with the question of whether they're a bank that builds software, or a software development company that specializes in banking.

Further reading
For more insights into software's encroachment into business,
we recommend Marc Andreessen's essay, Why Software Is Eating the World. →

The good news is that there's no need to reinvent the wheel when it comes to designing a software factory. There are plenty of proven architectural blueprints to choose from, including agile, SAFe, TOGAF and IT4IT, that can guide you toward industry best practices. Many of them make the same claim: that organizations don't need to apply their methodology by the book; they can adapt it to their own needs. Often, this is presented as an advantage – but let's explore why this might be too simplistic a view.

Avoiding the customization pitfall

Customization is great in theory. A business can adopt the parts of a software factory blueprint it likes, leave out the parts it doesn't, and alter anything it wants to. In practice, this is where problems can find a way in. Many organizations will customize a framework not to support what they actually need, but to fit the way they have always done things. Take it from us: it might seem like you're smoothing the transition in this way, but in fact it's more likely that you'll simply prevent the new way of working from actually working (or at least from doing so effectively).

In our example software factory, for instance, imagine a newly agile team or process that's surrounded by departments that still want to work in a more traditional way. (This is the equivalent of production in a goods factory adopting an on-demand strategy while sales and management continue to follow stock and production forecasts.) Customization seems like the solution: adapting the new methodology to fit the wider organizational pattern and using reports or spreadsheets to paper over the gaps. However, while it's not the case that this couldn't ever work, the likelihood is that it won't be as efficient as a fully aligned factory.

The better option is to avoid the pull of customization and go back to basics. Review the different methodologies available and choose the one that works best for your needs (remembering Conway's Law). Then stick to it as much as possible, and master the basics before you start customizing – not the other way around.

Why? Because you can only spend your time once. A framework doesn't dictate your vision and strategy, so avoid customizing the framework until you really understand the (dis) advantages of its components. Instead, use your available resources to define what make your business unique.

Learning from success stories

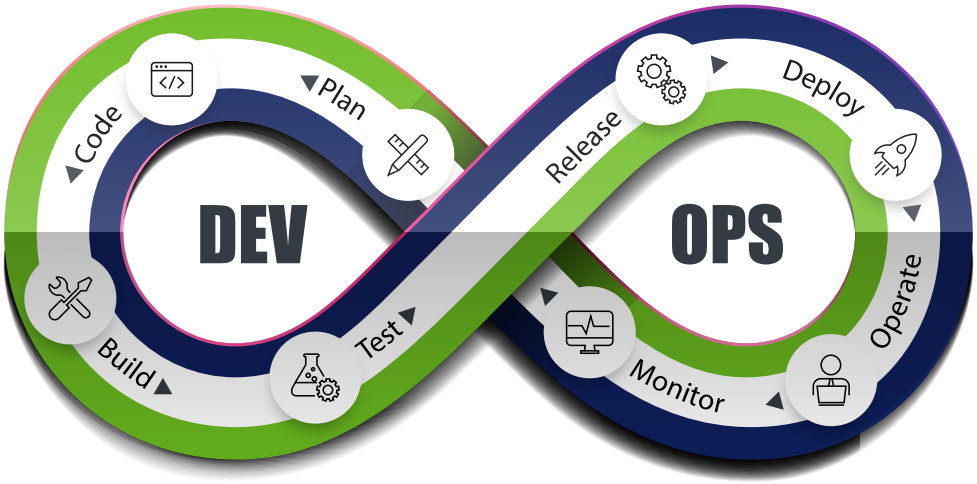
Having looked at what not to do, let's now consider some positive role models for our software factory. Spotify is a well-known example of a company that has made a big success out of a scaling up organizational agility – to the extent that it now has an entire approach named after it: the Spotify Model.

Further reading

For some agile inspiration, take a look at the Spotify founders' white paper,

Scaling Agile @ Spotify. →

It's also worth looking away from software to see what real-world factories can teach us. For electric vehicle manufacturer Tesla, the factory is the product. Each of the company's 'Gigafactories' is built as a more efficient iteration of its predecessor. New best practices are then retrofitted to older factories as an upgrade – putting the principle of continuous optimization into action.



In the software development world, this is known as refactoring: applying lessons learned and progressive insights to existing code. More than just a good idea, refactoring is a necessity to ensure the codebase can be maintained and/or extended in the future. Likewise, in a software factory, updates and upgrades are key to making sure its processes run at optimal efficiency in the long term.

(Keep) asking the right questions

When the time comes to start building (or renovating) a company's software factory, taking the first steps can be daunting. It's up to management to make the key decisions, right? Arguably, yes – but only to a certain extent. Consider Tesla: is its management responsible for deciding how its factories are built? Certainly, they would have the final say on what, where, and when. But the how is the responsibility of experts who know what's required to deliver the goods at the right speed and quality. The same applies in a software factory, where specialist engineers need to be involved in constructing the optimum architecture.

So, to ensure you land on the ideal design for your software factory, take a big-picture approach. Be sure to give some thought to a wide range of clarifying questions, such as:

- **What outputs do we need to build in the (near) future?**
- **What processes do we need to build them?**
- **How should we organize these processes and our related resources?**
- **Is there important maintenance due?**
- **Do we need new equipment or inputs to facilitate our new product line?**

And remember to keep asking. As with all things iterative, this process becomes easier the more you do it – and the more you do it, the more manageable (not to mention impactful) the necessary changes are.

Conclusion

Drawing a parallel with goods production is a useful way to better understand the need for efficient software development processes – and how to go about achieving operational excellence.

Golden rules for software factories

1. The factory's architecture is key to its success.
2. This architecture should be built to reflect the desired software architecture.
3. Both architectures need regular optimization: think Inspect, Iterate, Improve.
4. All this demands a shared vision and robust alignment across the entire organization.

Even when an organization employs the very best developers, testers, and other professionals, it will often find its hard work limited by a company structure that doesn't align with the outputs it needs to produce, or that doesn't allow access to the root cause of an inefficiency. Using a proven methodology that promotes regular maintenance can help, especially when it's embraced by the wider business.

There's a lot to be gained from continuous optimization in software development. In today's world, it's well worth paying regular visits to the factory floor – and staying tuned to the giveaway smells of inefficiency – so you can act to maintain the state-of-the-art operation your business needs. After all, it could be just a small tweak that makes a big difference to the performance of your software factory and, ultimately, the success of your product.

Contact us

At alfa1, our experts have toured plenty of software factories: the good, the bad, and the ugly. We know what works and what doesn't, and we're always excited to share what we've learned – so if you want to know how we can help refactor your software factory, don't hesitate to get in touch.

About the authors

Jeroen Benkhuijsen is a Java Solution Architect and Competence Manager at group9.
He advises clients on software solutions and the optimisation of all aspects of the software development process.
Internally, Jeroen is responsible for group9's competence policy.

Jasper Siegmund works as a Microsoft Solution architect for mStack.
As a consultant, he advises our clients and supports them with the design and implementation of Microsoft Azure-based software.
Azure is his cloud platform of choice, .NET his toolset.