Boost Your Digitalization: 8 capability areas to build

Jan Bosch

EXECUTIVE SUMMARY

The digital transformation of industry and society is a major shift that in many ways can be viewed as parallel to the industrial revolution. We are only in the early stages of fully exploiting the digital technologies that have come available to us over the last decades. These technologies include software, but also data and artificial intelligence.

As digitalization is so encompassing and multi-faceted, it changes is literally everything in the company as well as in the business ecosystem in which the company operates. Consequently, it is hard to know where to start and to prioritize change initiatives in a way that allows for achieving short term benefits while transitioning towards the long term vision.

In our research, we have identified 8 capability areas that companies need to build and develop, organized in 8 dimensions, i.e. business, architecture, process, organization, user innovation, technology innovation, automation and ecosystem. For each dimension, we discuss the key topics to be addressed.

The intent of this short book is to offer a map and a path for companies to successfully transform themselves into digital companies and to not only survive, but also thrive in a digital world. The journey is not easy and requires many sacrifices, but the alternative is much worse. So, it's important to get going now and not wait for a better time. As we all know, the best time to plant an olive tree was 40 years ago, the second best time is now. Let's go!

TABLE OF CONTENTS

DIGITALIZATION: 8 CAPABILITY AREAS TO BUILD

During the last weeks, I reflected on the research activities in the Software Center for which I am the director and I realized that everything we do with the partner companies and the universities is concerned with increasing the pace at which we are delivering new value to customers. Whereas most companies, a decade ago, relied on product generations to deliver new value to customers, over the last decade we have seen a shift to periodic software updates and now continuous deployment with very frequent updates to deliver new value. We are moving to a world where every system we use gets better all the time.

Although continuous deployment is still the challenge for many companies, it doesn't stop there. We now see that the next hills to take include conducting A/B testing using systems deployed in the field as well as (federated) reinforcement learning to allow systems to fully autonomously experiment with their own behavior to continuously improve the value experienced by customers.

Although everyone talks about digitalization as a goal in itself, in my view it is an enabler to achieve something else: a fundamental shift from a transactional business model and relationship with our customers to a continuous one. The transition from keeping products as static as possible once these leave the factory to continuously changing and improving already deployed systems requires a significant change across the company. Digitalization is such a hard challenge as it literally touches every function and capability in the organization, ranging from business models to deployment and from sales to finance. In order to survive and, preferably, thrive in this transition, companies need to develop new capabilities to capitalize on the new model of delivering value to customers.

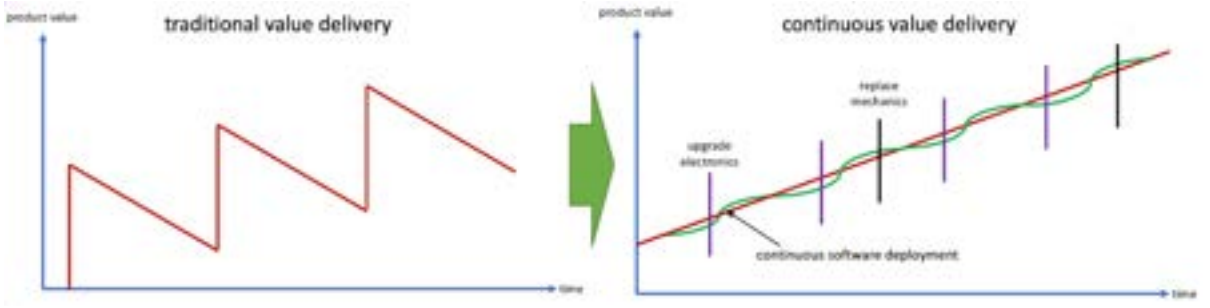


Figure: From traditional to continuous value delivery

In the figure above, the transition is illustrated graphically. In traditional value delivery, each product, once it leaves the factory, is basically frozen and deteriorates over time until the customer decides to replace it with a new product. That new product typically is a little better than the previous one, but it also will deteriorate over time and be replaced. In the continuous value delivery model, the value that the product, system or offering provides is continuously increasing. This occurs through continuous deployment of new software, but may also require periodic replacement of electronics and occasional replacement of mechanical parts. The system gets better every day I use it.

As alluded to in the title, successfully transitioning towards continuous value delivery requires organizations to build new capabilities across the company. Major changes are required in at least the following areas:

- **Business**: The shift from transactional to continuous value delivery has business strategy implications and changes the business model and the customer interface, including sales and support.
- **Architecture**: Both at the system and the software level, the architecture needs to be addressed to support a superset platform approach, modularization and instrumentation.
- **Process**: The ways of working in a continuous context require much faster cycles of iteration which affects our processes. Many tasks need to execute in parallel instead of sequentially, which of course is the nature of agile. However, it also requires a mature build and test infrastructure. Finally, the short cycles allow for data-driven ways of working that were not feasible earlier but these need to be capitalized upon.
- **Organization**: Traditional, functionally organized companies are great at efficiency of repeatable tasks in low change environments but very poor at responding quickly in high change environments. We need new ways to organize, new skills and different leadership styles.
- **User Innovation**: Especially in software-intensive industries, most innovation was traditionally of the sustaining, evolutionary type. In a digital transformation, we need radical innovation which works quite differently as we are inventing new ways of serving customers through new offerings and services.
- **Technology innovation**: The constant flow of data-driven technologies, including experimentation approaches such as A/B testing and machine- and deep-learning require us to invest in technology-driven innovation to complement user innovation.
- **Automation**: One of the sayings in agile is that when it hurts, you should do it often. Digitalization and DevOps create this reality and the consequence is a significant increase in the need for automation. The challenge is to find where it hurts and then automate in such a way that we keep relevant flexibility in the areas that matter.
- **Business ecosystem**: The superset platform, continuous value delivery and data from the field allows for a very different interaction with the business ecosystem around the company. By platformizing, we can open up to third parties to build extensions to the entire portfolio, creating multi-sided markets with network effects.

This may seem like a lot of take in, but it hammers home the challenge of digitalization and why so many companies struggle with the transformation. I see many companies take initiatives in one or a few areas and then ignore other important dimensions, resulting in failure, disappointment and loss of valuable resources. In the coming weeks, we are going to work our way through each of the capability areas and deepdive into each of the key capabilities in these areas.

Concluding, surviving and even thriving in a digital transformation changes the entire company. It requires building new capabilities in at least seven areas, i.e. business, architecture, process, organization, user and technology innovation, automation and ecosystems. Focusing on one and ignoring the others results in failure and disappointment. Instead, focus on the promise of continuous value delivery to customers and lasting, continuous relationship with your customers where everything gets better all the time. Who doesn’t want that?

BUSINESS DIMENSION

For all the talk about digitalization over the last decade or more, I still run into many people outside of software R&D who are convinced that digitalization is a technical problem that doesn't concern them. Often, it's viewed as an opportunity for improving the efficiency of existing processes and ways of working. The result is that the digitalization efforts in the company tend to address all the secondary aspects but conveniently ignore the core.

The essence of digitalization is the fundamental shift in value delivery to the customer. Moving from a transactional form to a continuous one. This includes traditional service approaches where, for instance, the traditional product is offered as a service or where other aspects of operating the product, such as maintenance, are provided as a service. However, we're predominantly looking for ways where we can continuously improve the value that we deliver to customers through the continuous deployment of new versions of software, data-driven insights and the use of artificial intelligence.

One core dimension of the digital transformation that needs to be addressed is the business. If we're unable to capture part of the continuously improving value to customers, we have no incentive to provide this value as we'll never be paid for it. For instance, automotive companies have, for the most part, connected products and could, for example, run A/B experiments to determine the optimal tuning of engines to maximize fuel efficiency. However, most of these companies lack a business model to monetize the improvements that they could provide. So, even though the customer would benefit, there's no incentive and, consequently, there's little tuning of engines after SOP (start of production).

The business dimension of the digital transformation has three main aspects to be addressed: the business model, the sales organization and customer support. To get incentivized, we have to be able to monetize the additional value we deliver to customers. Hence, the business model needs to change.

In most cases, I see companies initially pursue a two-pronged approach where the traditional transaction model is complemented with a continuous business model through which improvements in value are monetized. As an example, one company I work with uses performance contracts to share the value of any post-delivery improvement equally with the customer.

Once the continuous revenue becomes sufficiently large, many companies explore ways to shift revenue from the transactional to the continuous leg and potentially close the transactional revenue stream altogether. One challenge here is that the company cash flow is quite fundamentally disrupted as you now need to finance the equipment at the customer as there's no upfront payment anymore. In the long run, continuous business models are often more lucrative, but the short-term cash flow shortage needs to be managed.

The second aspect is the sales organization, which, in many companies, has for many years optimized itself along the lines of the old business model. This optimization includes the skills of sales reps, incentives and culture. Asking sales reps rooted in the old model to start selling services and continuous value delivery models without any changes to incentives and training is bound to fail. I know of at least five companies that had successful digital offerings that customers wanted and that R&D had prepared for delivery but that failed as sales was unable or unwilling to sell the offering.

I see three models being used in practice. First, companies try to adjust and train the existing sales organization to sell services in addition to products. Second, within the same organization, there are two groups of salespeople: those selling services and those selling products. Finally, the most extreme version is to create two completely separate business units where one develops and sells traditional products and the other develops and sells service offerings.

The third aspect is customer support. Traditionally, this department has quite a bit of time to build up experience and knowledge after the release of a new product generation. However, when adopting continuous business models and the associated continuous value delivery through, among others, DevOps, the lifetime of one version of the system is very limited and measured in weeks. Consequently, customer support has very little time to accumulate relevant knowledge and the risk is that customers end up without proper support. This is of course a problem for any company but even more so for companies using continuous business models as the switching cost for customers tends to be lower.

In addition, continuous deployment is an important enabler for A/B experimentation and other forms of testing in the field. This results in different customers using different versions of the software in the product, or different configurations in general, and customer support needs to know about the variants out in the field and their support implications. Building up the internal information channels can be a significant challenge as the involved departments traditionally had little to do with each other.

Digitalization isn't just a technical problem affecting R&D only but touches all functions in the company. The business side is one of the key dimensions and digitalization affects, among others, the business model, the sales organization and customer support. Failing to implement the relevant changes in these organizations will disturb the relationship with customers and put the entire digital transformation at risk. Remember, digitalization is, at its core, a fundamental shift in value delivery from transactional to continuous and this is a business challenge first.

BUSINESS: BUSINESS MODEL

Digitalization allows companies to fundamentally improve the way they deliver value to customers from transactional to continuous. Transactional value delivery is where a mostly physical product is sold as-is and then deteriorates over time until it's replaced with the next product. Continuous value delivery is concerned with using the installed base as an enabler to continuously deliver new value to customers through various ways including DevOps, DataOps, MLOps and A/B testing.

Although this may be obvious to many of you, many companies struggle with the fact that changing the way value is delivered necessitates a review of the business model used to capture part of that value for the company. The danger is that we monetize the offering using the traditional, transactional business model but then commit to a continuous cost structure where we're expected to conduct R&D and other efforts to deliver continuous value over the lifetime of the product.

The only way in which a continuous cost structure can be combined with a transactional revenue structure is when you can charge a significant premium for the product. Apple mobile phones are a good example of this model as you, after you've bought the phone, basically receive product updates for free. However, if you're unable to charge the premium due to competitive pressures, you're painting yourself into a corner.

When you can't charge a premium on the product, you basically have three options: don't deliver continuous value, complement your existing transactional business model with a continuous business model or entirely replace your business model with a continuous one.

Deciding to not deliver continuous value is always an option in the short term, but customer expectations are rapidly shifting towards that model. For instance, my car, a German brand, provides lane-keeping and adaptive cruise control. However, different from Tesla, where new software is delivered periodically, I'm stuck with the same quirks and, frankly, sub-par functionality. I expect the

products I use to get better and I get really annoyed if they don't. And I believe I'm far from the only one.

Complementing a transactional business model with a continuous one is what many companies I work with are opting for: the idea to create a secondary service revenue stream to complement the primary transactional revenue stream is quite appealing as it's viewed as low risk – it's optional for customers and doesn't put the current business model in jeopardy. The challenge in this model is, however, that whenever there's a conflict between the product and the service, the product wins as it tends to generate a vast surplus of revenue.

The more radical approach is to shift all revenue from transactional to continuous. Many years ago, Rolls Royce, the jet engine company, adopted their "power by the hour" model where you pay for use of the engine, rather than the engine itself. I believe you can't even buy a jet engine as a product anymore.

Continuous business models aren't all born equal. In addition to usage-based monetization, there are subscription models, which are concerned with access, as well as performance-based models, where monetization is based on the value measurably created at the customer. Some companies I work with use DevOps to continuously experiment with ways to improve certain KPIs and have contracts with their customers to equally share the created value between them. This is but one model; there are many more to consider.

Shifting from a transactional model of value delivery to a continuous one calls for a continuous cost structure to deliver new value. In turn, this requires a review of the business model or models used by the company to compensate for this. This can result in charging premium prices for the product, complementing the transactional business model with continuous revenue, eg through services, or replacing the existing business model with a continuous one. Which alternative to choose is highly context dependent, but ignoring the ability to deliver continuous value and the emerging cost structure if you do are non-starters, to begin with. In the end, although the purpose of a company shouldn't be about making money, we still need to make money to have a viable business. Remember, as Bing Gordon said: "In any business that grows big on one business model, transitions can throw everything in the air."

BUSINESS: SALES

For most of my life, I've worked in engineering. I worked as a programmer during my university studies. Built my own compilers while conducting my PhD research as I looked into alternative object-oriented programming languages. Worked in industry as a vice president on the engineering side and, of course, I published hundreds of papers on technical topics ranging from software architecture to platforms and from data pipelines to federated learning.

Over the last decade, however, I've increasingly started to realize the importance of sales. For many people, this is a slightly dirty word. Images come to mind of sleazy used-car salesmen and annoying people ringing the doorbell or phone at the most inopportune times to try to sell you stuff you don't need.

In practice, however, we're all salespeople in the work we do. Whether you're trying to convince your colleagues to change a process or ways of working or submitting a research proposal at a funding agency, you're doing sales and it helps to think of it that way. Sales is about building empathy with customers, deeply understanding what drives and is important for them and then formulating an offering that meets that need.

Since all startups that I've been involved in and that didn't succeed failed due to lack of sales, I'm highly interested in and aware of the importance of sales. And even if we may all be selling in one way or another, it's the dedicated sales professionals who convert meeting customer needs with your offering into cold, hard cash.

The challenge for sales as a company digitalizes is that the value offering changes from a transactional one-off sale to selling an offering that's continuously monetized. There are at least three main changes that sales needs to go through for the company to succeed. First, the customer relationship moves from a transactional to a continuous one. So, rather than getting the customer to sign at the dotted line and

moving on to the next target, sales is now about building, maintaining and growing the relationship with the customer.

Second, in my experience, many salespeople are very competitive and get a kick out of closing deals, getting their bonus and seeing it accumulate over the quarter. In a continuous model with the customer, the benefit tends to be shaped very differently as the initial revenue for the company is much lower and the associated bonus is different as well. In my work with a variety of companies, I've experienced quite a few cases where engineering was able to build the innovative offering, customers wanted the offering, but sales failed to sell it and hence the innovation failed. One of the key factors is that the incentive system for sales staff needs to change for it to make sense for them to spend time on selling services.

Third, for some reason, I meet quite a few people who believe that simply because the offering is more digital and continuous the sales process will magically become inbound and all driven through digital means as well. In my experience, that only happens, if at all, once you've built up a brand and scale that result in a very high degree of awareness in the market. Instead, the continuous relationship with the customer requires even more human interaction as the level of trust required is higher. It's no longer a date, but you're actually going steady! Consequently, the focus should be on a land-and-expand strategy rather than trying to achieve the highest value in the first sale.

Digitalization touches all functions in the company, but the business dimension is one of the most important ones. With the business model being reinvented, sales also needs to reinvent itself. This requires changing the way we build, maintain and evolve relationships with customers, adjusting the incentive system for salespeople to ensure that they benefit from selling the new digital offerings and adopting a land-and-expand strategy. Cross-sell and up-sell become much more important in a digital transformation. For many salespeople, this is quite a shift from how they've operated in the past, but as Ginny Rometti, the CEO of IBM, at some point quipped: growth and comfort don't coexist.

BUSINESS: CUSTOMER SUPPORT

So far, the posts in this series have stressed that the continuous value delivery to customers, rather than the transactional model, affects all parts of the company. One of the areas often ignored or seen too late as playing an important role is customer support. In many companies, this team is assumed to already have a continuous relationship with customers. In practice, however, most customers may have some interaction when initially starting to use the product and then again when the product reaches end of life and starts to show failures.

When transitioning to continuous value delivery, customer support runs into at least two main challenges. First, the team needs to build up knowledge about the latest software release to the field much quicker than traditionally. Whereas customer support used to have weeks if not months for that, when new software is released every two to four weeks, the knowledge about it needs to be built up in hours and days. Second, as not all customers upgrade their software at the same time, it can easily become difficult to know which version of the software the customer is experiencing issues with and how to best provide support.

The result is that customer support needs to change in three important ways: deepened customer empathy, accelerated knowledge acquisition and proactive support. First, as we build a continuous, rather than a transactional, relationship with customers, customer support has to expand its scope of concern and understand much better the context in which the customer is using the solution. The focus of the customer has always been to achieve certain outcomes and your product has simply been a means to that end. Whereas customer support could traditionally focus on the functionality of the product itself, in a continuous relationship, the team needs to focus on the outcomes the customer is looking to accomplish.

Second, as is obvious from the discussed challenges, if new software is deployed every two to four weeks, the knowledge acquisition by customer support for each release needs to accelerate significantly.

Often this requires customer support to become part of the overall DevOps cycle so that they can predict the most likely customer support issues that will come up and prepare for that.

Third, as the product is now connected, we can collect data and consequently monitor the performance of the product. This means that we can implement the biggest shift of all: from reactive customer support where we wait for the customer to call to proactive customer support where we fix issues before the customer is even aware of them. You can then either inform the customer post-hoc or keep things quiet.

In many companies, customer support is viewed as a bit of necessary evil and a cost center that should be optimized for a minimal total cost of ownership. The digital transformation, however, changes the role and nature of customer support quite fundamentally. As a consequence, rather than outsourcing the function to a call center company or something similar, it now becomes important to integrate it into the DevOps lifecycle and enable proactive customer support. In a continuous value delivery model, customer support suddenly becomes your primary interface to the customer and that interface should be as strong and supportive as possible. In the end, as the saying goes, good customer service costs less than bad customer service. That's even more true in a digital world.

ARCHITECTURE DIMENSION

While the last posts were about the business dimension of digitalization, there obviously is a technology side to it as well. Digitalization requires the company to build new technical capabilities, specifically around the use of data and artificial intelligence, but the primary challenge is, in my experience, concerned with system and software architecture.

The system and software architectures embed the key decisions about the business of the organization and how we deliver value to customers. And although the architecture isn't immutable, it's often more expensive to change the architecture than other aspects of a system, such as features and interfaces. Also, as few of us ever have the opportunity to start from a greenfield situation, we'll need to evolve the current architecture to facilitate the digital transformation. Architecture transformation tends to be slow and time consuming as many architecture design decisions have implications throughout the system and removing these is very demanding.

Especially in embedded or cyber-physical systems, the traditional approach was to force the software to follow the mechanical and electronics lifecycle. Similar to leaving mechanics and electronics unchanged after the start of production, the software is typically updated as little as possible too. This allows architects to build all kinds of dependencies in the architecture as these will incur a quality assurance cost only once.

With digitalization, we seek to improve the delivered value continuously through digital means, including updating the software and AI models as well as using data analytics to optimize the performance of the system. This has implications on the system architecture as it requires the mechanical and electronics parts to be designed to optimally support continuously improving value delivery. Suddenly, the atoms are in service of the bits instead of the other way around.

The implications on the architecture are, at least, threefold: platformization, modularization and instrumentation. First, the transition from delivering software for products once or infrequently during their lifetime to continuous updates easily becomes prohibitively expensive in case each product or each

customer-specific instantiation of a product is treated as a separate, unique entity. Instead, we need to adopt a “superset platform” approach where all products and all product variants are configurations of the platform. All functionality delivered to customers lives in this one platform and all additions are made to the platform. This allows for a much higher degree of automation of quality assurance and deployment, but it has significant architectural implications. The architecture needs to support the superset of functional and quality requirements as well as a mature approach to managing variability.

Second, as most systems, especially in embedded contexts, are large and complex, we need to shift away from the traditional approach of shutting down the system, updating the entire software and restarting the system. In the case of DevOps, this causes too much system downtime and will lead to customers resisting upgrading their systems regularly. Instead, we need to support the independent deployment of components. This requires a modularization of the architecture to allow the system to function properly even if the components that make up the system are of different generations and versions. This in turn requires defensive approaches for component interfaces as well as maintaining multiple interfaces for the component to support different system configurations.

Finally, one of the key elements of digitalization is the use of data from the field for analytics and machine learning, both centrally and in the deployed systems. This requires the architecture to support easy instrumentation of components, features and interfaces. As it’s impossible to predict what the company might want to collect in terms of data, it’s critical to be able to easily add various types of instrumentation to the system as part of the DevOps cycle. In addition, as the amount of data generated in various parts of the system might be quite significant, some of the processing of this data needs to be done locally to decrease dependency on a centralized system.

Digitalization also has implications for the system and software architectures. The transition to continuous value delivery requires, especially for embedded systems, a kind of Copernican revolution in the system architecture where the atoms need to support the bits, rather than the other way around. This requires three main architecture changes: adopting a superset platform approach, significantly strengthened modularization of the system and mechanisms for easy incorporation of instrumentation as well as data processing functionality. To paraphrase Winston Churchill, we shape our architectures, but then our architectures shape us. So, design the architecture supporting your digital transformation carefully.

ARCHITECTURE: SUPERSET PLATFORM

Sometime last year, I had a discussion with senior leaders of an automotive tier-1 supplier. I explained to them that if we're serious about moving towards DevOps in automotive, their customer-project-centric approach would have to change. A typical car model is developed for three to four years, then manufactured for around seven years and the OEM is obliged to provide software support for it at least eight years after the last car runs of the production line. Assuming the tier-1 supplier supports 10 OEMs and each OEM has 10 car models in production, the traditional way of working would mean the company would need 100 teams to continuously evolve and validate the software for all car models in the field for more than 15 years per model.

This situation is the case for every company with a portfolio consisting of a range of software-intensive products. Even if you could organize the work such that each product team pushes out software updates every two to four weeks, the cost of this approach would be so prohibitively high that no business model can support this.

The solution to this conundrum is, in my view, the superset platform. This is built on a single codebase that contains the superset of all functionality in the product portfolio. The software for each product is simply a configuration of the code in the single codebase, rather than its own codebase. In addition, the platform is complemented with a continuous integration and test infrastructure that ensures that each product in the portfolio is always at production quality.

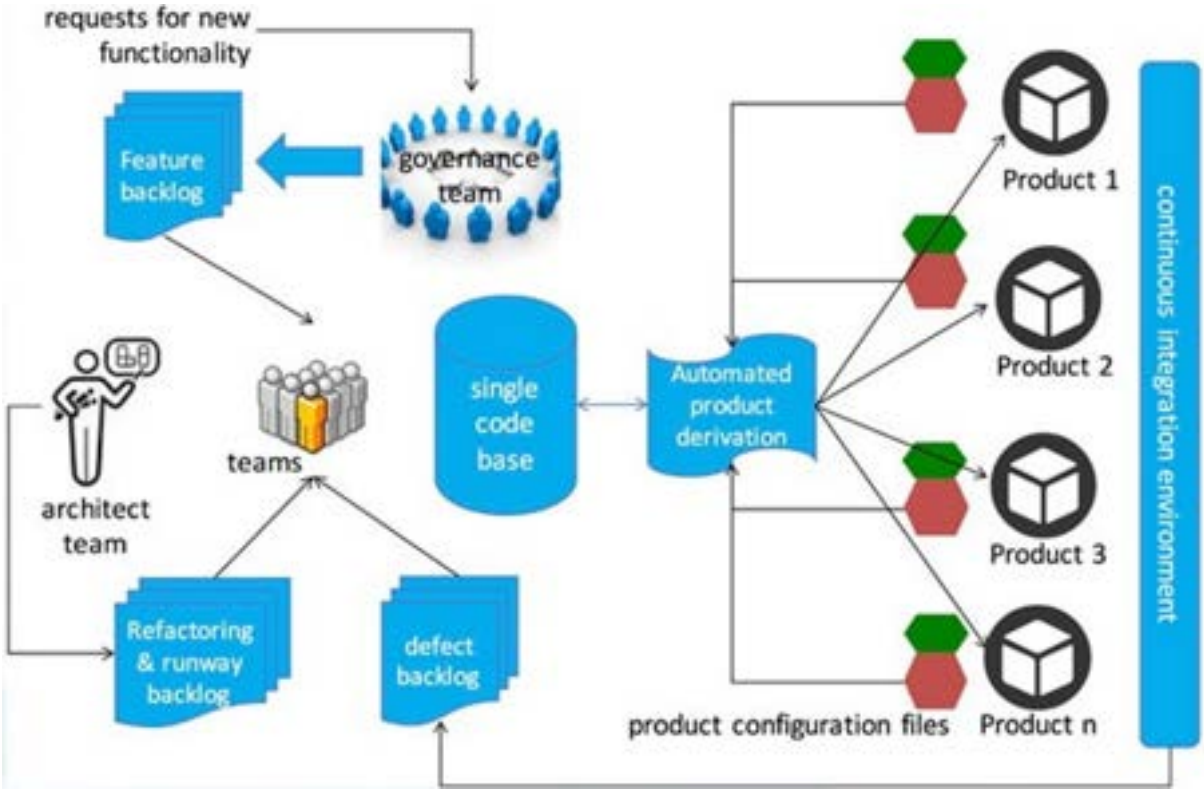


Figure: Illustrating the superset platform

The advantages of the superset platform approach aren't limited to better support for a DevOps setup. One major benefit is that it allows for much better governance of resources across the portfolio. In the situation where each product has its own associated R&D team, each team will focus on delivering the most value for its product. However, from a portfolio perspective, this easily results in a situation where one team lacks the bandwidth to do important work for its product while teams associated with other products are busy with less important work. Pooling all the resources in one organization supporting the superset platform allows for the allocation of resources in line with the company's top-level priorities. A second benefit is that features developed for one product can easily be provided in other products in the portfolio as well.

As with everything in life, the superset platform also comes with a few challenges. The primary challenge for most companies is to develop proper mechanisms for managing the variability in the platform to allow for the proper configuration of products. However, there are well-defined solutions and tools available for addressing this. Second, the overall platform will be larger and more complex than the codebase for each individual product, meaning that you'll need to ensure stronger architectural oversight. In my experience, however, the advantages of the superset-platform approach, especially in a DevOps context, significantly outweigh the disadvantages.

Adopting continuous value delivery as part of the digital transformation requires us to adopt DevOps as the vehicle for delivering software-based value. When adopting DevOps, a traditional product-centric approach rapidly becomes infeasible. Instead, we need to adopt a superset-platform approach where all products are automatically derived from a single, common codebase and automatically tested to allow for low-effort continuous deployment. It's yet another perfect example of one of the key tenets of Agile: if it hurts, do it often!

ARCHITECTURE: MODULARIZATION

One of the concepts I think is poorly understood by many is the notion of modularization. Generally, most people will consider a more modular system as preferable to a less modular system. As a consequence, modularity is viewed as positive and generally used as an alternative word for "good."

My concern with this view is that it ignores a few realities about systems engineering. The first is that modularity typically has a downside in terms of reduced efficiency and increased resource usage. In a more integrated system, different parts can access each other more directly and with less overhead. This is why in traditional systems engineering, modularity is sacrificed to increase efficiency so that components with lower performance and cost can be used. So, modularity has a price many fail to recognize.

Second, for any architecture, the dimension along which you decide to break the system into its primary components removes modularity along other dimensions. During development and maintenance, a rule of thumb in software architecture is that one change request should lead to change in, preferably, one component. As an example, many know that the primary components of a compiler include a lexer, a parser and a code generator. This is a logical decomposition as it allows each component to focus on its particular responsibility. However, in the case of an extensible programming language where new syntax elements can be added over time, this architecture is very non-modular as every new syntax element will require changes to the lexer, the parser and the code generator. So, every change request causes changes in every component of the system. In short, a system is modular in response to a specific set of expected changes and by choosing an architecture that can be considered modular for these expected changes, the architecture is, as a consequence, non-modular for other types of changes.

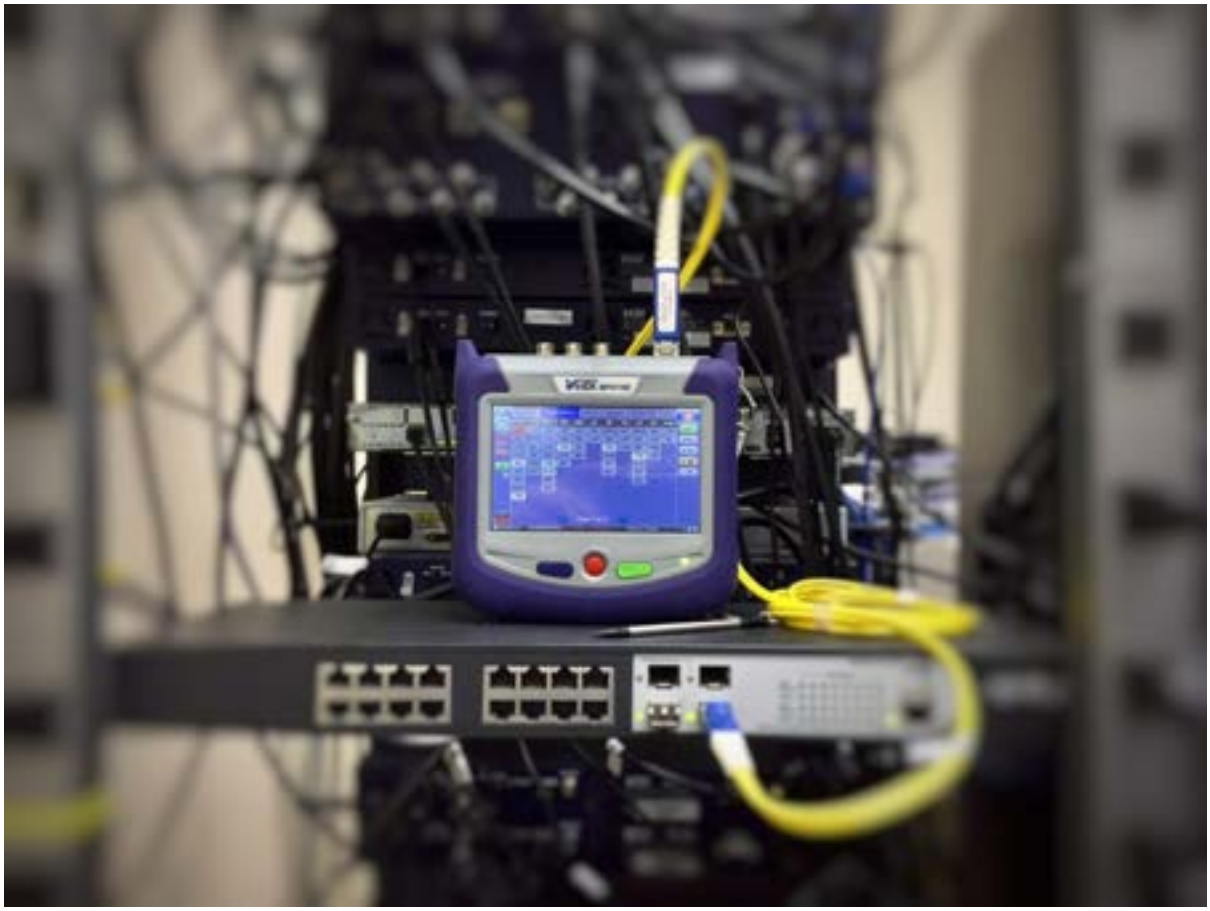
This brings me to the third misconception around modularity: the assumption that modularization means the same before and after a digital transformation. Traditional systems often are modular during development time and integrated during execution time. In embedded systems, in the cases where new

software is deployed to systems in the field, there typically is an “image” that replaces all software in the system. As a consequence, an upgrade often is quite disruptive in that the system has to go offline, enter some special state required for updating, go through a sometimes lengthy update process, restart and run an extensive self-test before becoming available for operations again.

In a world where we want to continuously deliver new value to customers, the traditional upgrade process is too cumbersome and disruptive to conduct frequently. Instead, we need to support the independent deployment of components without disrupting the system operations. Typically, this requires the old and new version of a component to co-exist in the system for a while as traffic is routed to the new version and the old version concludes its ongoing processing. Once the old version is dormant, it can be removed.

This form of modularity requires capabilities from the architecture that often aren’t present in systems. It requires careful modularization of the architecture, the introduction of infrastructure to manage the co-existence of multiple versions of components, run-time testing of functionality without affecting operations, as well as instrumentation to detect anomalous behavior and perform automated responses, such as roll-back.

Modularity and modularization are concepts that tend to be poorly understood in general. As we’re going through a digital transformation, the meaning of these concepts changes and expands to include the post-deployment stage of systems. This has several architectural implications, including changes to the principles driving architectural modularization, infrastructure to support seamless run-time updates and mechanisms to detect and address anomalous behavior. This is a lot of work and comes with a fair share of risks and challenges, but what’s the alternative? As George Westerman from MIT said: “When digital transformation is done right, it’s like a caterpillar turning into a butterfly, but when done wrong, all you have is a really fast caterpillar.”



ARCHITECTURE: INSTRUMENTATION

Even if many of us conceptually agree with the notion of data-driven decision-making, it's my experience that many still fall back into old-fashioned opinion-based decision-making. These opinions are formed by experience and for most of history, making decisions based on experience was a great principle as the pace of change was very slow.

The challenge with digitalization is that it represents a radical rather than a gradual change. As a consequence, things that were true in the old world are no longer true after a digital transformation. This is critically important as it has important implications for experience: instead of being valuable, it likely becomes a hindrance and something that holds organizations back. The capability that companies relied on for years is suddenly problematic.

The point is not that all experience has become useless, but rather that we don't know which aspects of our earlier experience are still valid after a digital transformation. So, the point is that we need to operate with a healthy dose of skepticism and follow the adage from Edwards Deming: in God we trust; all others must bring data.

The data is where, in my experience, the challenge starts in most companies. It often isn't available, either because it isn't collected, it's hard to access or there are data quality issues. It may even be the case that you have the data, but people refuse to believe and use it as it conflicts with their assumptions about the world.

The starting point is to have the data available and this requires instrumentation of systems in the field. Instrumentation isn't a new topic as most companies have some form of data generation in their

products, but the data typically is concerned with quality assurance and, to some extent, performance. Few of the companies I work with have any insight into the actual usage of their systems and can't even answer questions about feature usage and how their systems deliver value to their customers.

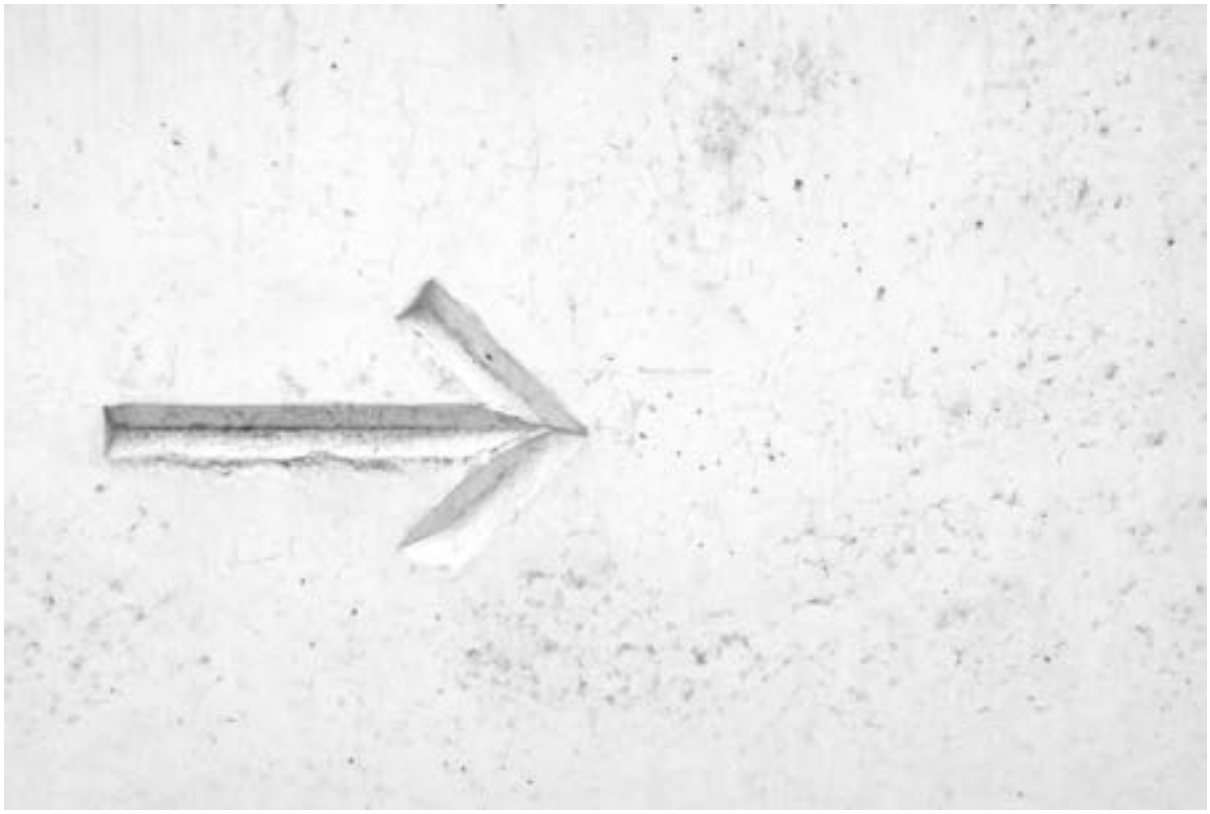
So, we need to rethink the way we instrument our systems. The existing quality and defect detection instrumentation should still be there, but we need to generate data that we can actually use for diagnostics and machine learning. In general, it requires that the data has more contextual information associated with it.

For instance, if you're looking to improve the fuel consumption of an engine and you only collect real-time fuel consumption data, you're not going to learn under what circumstances this consumption is high or low. You at least need to collect the requested power output and, likely, the gear the vehicle is in as well as the RPM to start to identify the situations where the fuel consumption is significantly higher than the requested power. With the combined information, we can start to develop hypotheses for testing in the field to improve the intended outcome.

The system architecture has to support instrumentation of functionality with as little effort as possible. This typically calls for architectural patterns where hooks are available to attach instrumentation functionality. In addition, as the total amount of raw data generated by systems easily becomes very large, forms of data processing to reduce volume and identify the most relevant parts are often required as well. This processing is preferably done close to the point where the data is collected or at points where multiple data streams come together. The architecture should, preferably, also allow for data processing functionality to be dynamically added to the system.

Especially in embedded systems, we tend to squeeze resources as much as possible to save costs. In this case, however, we need headroom in the system to allow for additional computation concerning data collection and processing. Also, we either need to include all data collection and processing into the system from the start with the functionality switched off until specific data is required or we need to support continuous deployment to allow for the addition of data-related functionality through subsequent software releases.

Digitalization requires data-driven decision-making as we need to be less dependent on opinions. This requires us to instrument our systems in the field so that we can collect the data required for decision-making. The system architecture should allow for easy incorporation of data collection and processing functionality so that we can answer relevant questions about system performance and customer behavior as well as experiment with alternative ways of realizing that functionality. Architecture is the physical incarnation of your real business strategy, so make sure you design it accordingly!



PROCESS DIMENSION

Wherever Agile principles and practices took a foothold in industry, the word “process” became anathema. In many ways, this was a healthy reaction to the CMMI approach to software engineering that preceded it and that was extremely focused on process. It turns out that in addition to process, we actually need architecture, technology, teams, commitments, retrospectives and other practices.

The challenge is that some in the Agile community threw out the baby with the bathwater and declared everything process as bad, old fashioned and constraining. Although this works well at the team level, once we go beyond that level, we need coordination mechanisms to ensure that we achieve the outcomes that we’re looking for at the organizational level. Typically, a customer expects the results from more than one team to be integrated into one offering.

In many organizations, this translated into a situation where Agile practices were used at the team level while above that level, everything stayed with the traditional waterfall-style processes. Even if the intention in approaches like SAFe was to scale Agile to the organizational level, the result in many companies is that we rehash the old ways of working into a new terminology, but in practice, nothing changes.

As we’re digitalizing our product portfolio, the consequence is much faster cycles of value delivery. This value isn’t just delivered through individual agile software teams but through the software R&D department as a whole as well as through mechanics and electronics. So, the first change we need is for the company to become agile for real, meaning that every function, department and team needs to operate based on Agile principles.

This doesn’t mean that mechanics and electronics R&D need to introduce two-week sprints and deliver value at that rate, but it does mean that they need to operate in much faster cycles than their traditional

ways of working. And I know of several companies where these groups adopted a sprint model and significantly benefited from it.

Of course, it doesn't stop at R&D; it affects the entire company. In the end, the goal is business agility, ie the company's ability to rapidly respond to changes in customer needs. This also includes sales, customer support and general management.

A second aspect concerns the traditional way of thinking about process as humans following process steps defined in some process specification. In practice, however, we can automate many steps so that humans don't need to bother with knowing all the ins and outs of the processes followed in the organization. Instead, a few experts can define the automation required for ensuring process compliance and the rest of the organization can, by and large, be blissfully unaware.

There are many examples from the realm of robotic process automation (RPA), but an illustrative one is the build and test infrastructure that digital companies need. As we push out updates to systems in the field frequently, we need to establish that these systems will function appropriately. This requires a build and test infrastructure that allows us to catch the vast majority of issues before they reach the customer. This infrastructure, in practice, encodes and automates the quality assurance processes used in the organization. Even if we may still want a human to do a final check of the resulting release, automated testing can be run much more frequently and with much higher consistency.

As a third aspect, we need to adopt data-driven ways of working. Most companies will claim they're data-driven, but in practice, every time the data collides with their beliefs, they reinterpret the data so that they don't have to change those beliefs. However, as there's much more data available due to the constant connection to systems in the field as well as with our customers, we need to use that data for decision-making and ways of working in general. Especially disciplines in the organization that traditionally have had very long feedback loops to the field need to adjust their ways of working significantly to incorporate data-driven practices.

Although traditional Agile declared everything process as anathema, the reality is that coordination across teams, disciplines, functions and departments in a large-scale organization requires processes. Digital transformation means that many tasks that traditionally were done sequentially now need to execute in parallel. In addition, the constant flow of value to the field requires significant changes to the ways of working across the company. This requires the company to adopt Agile for real, automate processes, especially around quality assurance, and start using the available data. As the saying goes: if you do what you've always done, you'll get what you've always gotten. And as we want something different, we have to change our processes and ways of working too!

PROCESS: AGILE FOR REAL

Every company I meet claims that they're agile. When I push a little bit and ask for details, typically the conclusion is that this is only true at the team level. The rest of the organization is stuck in traditional, slow cycles. The defense that I get is that the company is fast enough in responding to changing customer needs as they would have gone out of business otherwise.

To me, this is a typical case of "what got us here won't get us there." When reflecting on how we deliver value to customers, we can see a continuous acceleration of value delivery cycles. Initially, most companies deliver value through product generations. Every few years, the next version of the product comes out and people update from the previous version to the new one. Especially in software, but also for other technologies, the yearly update has become the norm in several industries, especially in IT. Now, we're moving towards DevOps, DataOps and MLOps and we deliver value every 2-4 weeks.

Of course, the speed-up of value delivery won't stop there. Especially in the online world, techniques such as A/B testing are used to even more quickly deliver value to customers. And, as a next step, some companies (and researchers such as in my group) are experimenting with techniques like reinforcement learning where systems fully autonomously experiment with their own behavior to optimize their performance.

To survive and thrive in a world where customers expect continuous delivery of new value, it's necessary that software R&D teams are agile, but that's far from sufficient. In this world, every function, department and team needs to operate based on Agile principles. In practice, the functions that struggle the most with the transition to a continuous value delivery are sales, customer support and mechanics and electronics R&D. We've already covered sales and customer support in earlier posts in this series.

Traditionally, the R&D teams concerned with mechanics and electronics often have a SOP (start of production) mindset. This means that they'll be focused on working towards a specific date, typically

rather far out into the future. At this date, everything needs to be in place, tested and guaranteed to work as it should as changes post the SOP point tend to be prohibitively expensive.

The interesting observation is that there's a fundamental shift between traditional and digital companies: for the former, everything stops at SOP whereas for the latter, everything starts there. Once we start production, we can get products in the hands of customers and only then, the data starts to flow that allows us to start the continuous value delivery.

Several of the companies I work with have changed the R&D process from a product and SOP focus to a stream model where each discipline develops a constant flow of improvements to the subsystems or components that it provides to the product portfolio. Any specific product is then a composition of the selected variant for each component and subsystem and almost a 'by product' of the stream-driven R&D process.

This approach supports the notion of continuous value delivery as the new versions of components and subsystems may also be available to systems already out in the field at customers. So, we can deliver additional value through new and improved mechanical and electronic components as well.

The main precondition for this to work, though, is a strong system architecture discipline in the company that embraces the digitalization principles and enforces backward compatibility not just for software but also for mechanical and electronics components. And, of course, it calls for a well-managed evolution of interfaces where the old ones are no longer fit for purpose.

Most companies claim they're agile, but in practice, that's only true for the software R&D teams. The other functions and departments are still using traditional, slow processes. Digitalization requires everyone in the company to adopt Agile principles. The incarnation of Agile may look different, but the overall goal is to enable the continuous delivery of new value to customers through all means possible. Digitalization requires Agile for real!

PROCESS: BUILD AND TEST INFRASTRUCTURE

The traditional way of software development is to first agree on what to build, then build the functionality and finally commence testing and fixing defects until the quality is at an acceptable level. As anyone who has spent any time in this field knows, no software is ever flawless, but the concept is to get the critical and major defects removed and bring the level of minor defects down to a reasonable number.

As part of the continuous value delivery to customers, we need to push out updates to systems in the field very regularly, ie at least every four weeks but potentially much more frequently. To be able to do so without causing major quality issues requires a different approach to system quality. The basic principle is that the software, as well as the system as a whole, is always at production quality and we never allow the quality to drop below that level.

To achieve this, for every check-in of new code to the source code control system, we need to verify that it doesn't cause any quality concerns. This requires that we build the new executable and then test it to ensure that no defects were introduced, meaning that the preexisting functionality still works as intended and that the new code works as intended.

Of course, for any system of non-trivial complexity, the time it takes to build and test it is longer than the average time between check-ins by the different teams and engineers on these teams. In response, most companies build up a staged model where each check-in is tested to a limited extent to ensure that the most obvious mistakes aren't present. Then, subsequent testing activities will combine multiple new contributions and test the system including these.

A useful way to visualize the complete set of testing activities between individual engineers checking in code and releasing it to the customer is the Continuous Integration Visualization Technique (CIVIT). In the example below, each box indicates one testing activity. Here, after acceptance testing (the

leftmost column of activities), testing is performed every hour, every day and every week. Of course, the test suite for the one-hour test is much smaller than the daily and weekly ones, but the idea is to continuously build up confidence in the quality of the code. In this example, the release organization still conducts separate, manual testing before releasing the software, but depending on the criticality of the system, it's also possible to automatically release software upon it successfully passing the most elaborate test activity.

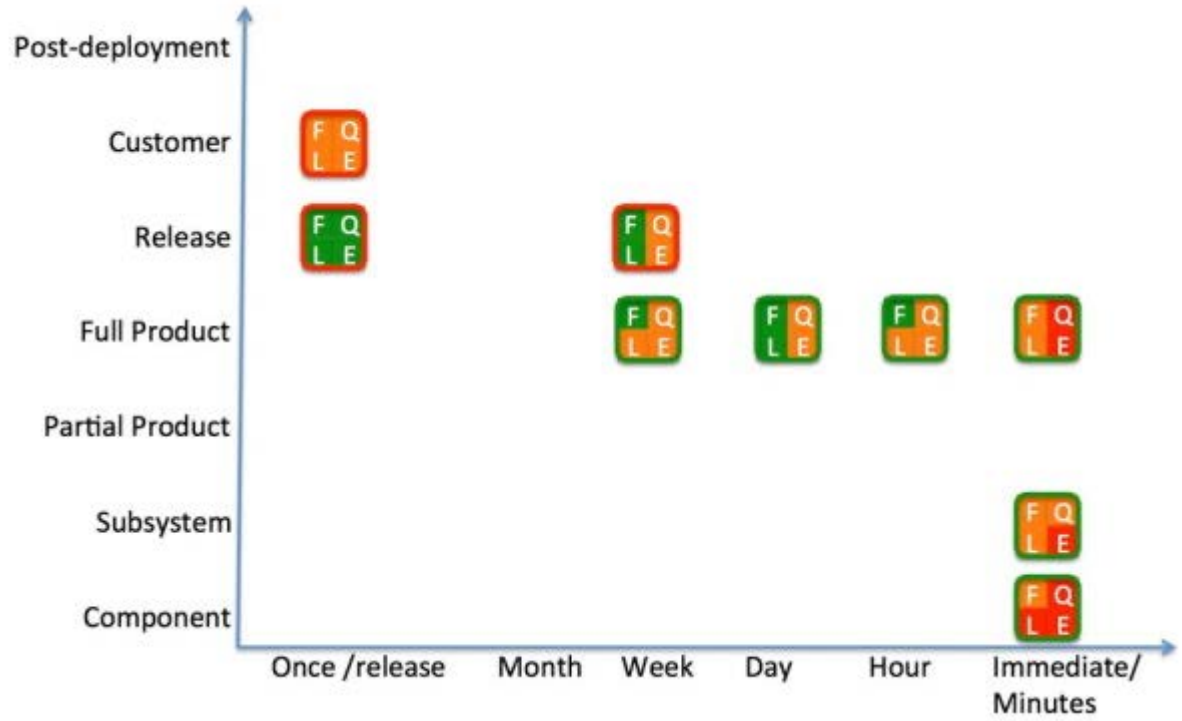


Figure: An example CIVIT model. The colors in the boxes indicate the test coverage for functional requirements (F), quality requirements (Q), legacy functionality (L) and edge cases (E), ranging from no testing (red), less than 30 percent (orange) to full coverage (green). The colors of the borders around the boxes indicate the level of test automation, from none (red), less than 30 percent (orange) to full (green).

One aspect not shown is the root-cause analysis process included in this way of working. Every defect that slips through to the field is analyzed from the perspective of improving the test suite to ensure that this defect or similar ones will be caught going forward. Working in this way allows companies to transition from a situation where they're managing hundreds or even thousands of known defects to a situation where the number of known defects in the field is in the single digits.

The second process that needs to be instantiated in this context is the maintenance of the test suites. This includes maintaining traceability between requirements and test cases, assigning test cases to the best suite (run more or less frequently), removing obsolete and duplicate test cases and addressing flaky tests.

Digitalization implies more frequent, if not continuous, delivery of value to customers. The main carrier of value is software, resulting in DevOps or continuous deployment. To ensure quality, we need to establish and maintain a build and test infrastructure that limits the number of quality issues that slip through to the field. This requires not only the infrastructure itself but also supporting processes such as root-cause analysis and test case maintenance. As John Ruskin said: "Quality is never an accident. It's always the result of intelligent effort."

PROCESS: DATA-DRIVEN WAYS OF WORKING

There's more data in the world than ever before. According to some sources, the world stores over 50 zettabytes of data. A zettabyte is 2 to the power of 70, or one billion terabytes. So, one would assume that the entire world has moved to data-driven ways of working.

As we all know, this is far from the truth. The vast majority of decisions in the world are based on opinions, potentially outdated beliefs, expired experiences and politics. In many cases, data is used as a mechanism to reinforce the points one is looking to make to convince others and there's little interest in starting from the beginner's mind as advocated in Zen.

Although it's easy to blame ill-intended leaders, the fact is that humans are simply wired to take mental shortcuts as a resource efficiency strategy. Much of our survival over the millennia depended on taking immediate, almost instinctive action in response to threats. Those deeply ingrained behaviors are as much present in modern humans as they were on the African savanna. Anyone who has been in a contentious meeting where things turned from factual to personal has seen evidence of that behavior and experienced it first-hand.

As humans, we're wired to ignore 99 percent or more of everything that reaches our senses. This causes a strong information bias as we typically note that which we agree with and ignore, even subconsciously, that which doesn't match our beliefs. As an example, in some of the company boards I've been part of, there's a very strong tendency towards groupthink. Humans want to be liked and respected by each other and therefore disagreement is often suppressed. I then take it upon myself to be the dissenting voice to avoid groupthink, but I've realized that this isn't to my advantage. Although others will treat me with respect, it doesn't help me build rapport with my fellow board members.

Because we're ill-designed to employ data-driven practices, we need to put mechanisms in place to ensure that we operate based on data instead of opinions, beliefs and politics. In my view, there are at least four mechanisms to consider. First, we need to ensure the availability of data. There are numerous cases where companies were drowning in data except for the data needed for a business-critical decision. This means that we need to be very careful and considerate in the type of data we collect and start collecting it long before we see its use become prevalent. As discussed in an earlier post, we need to architect our offering portfolio to make it as easy as possible to add instrumentation to systems in order to start collecting data that we didn't predict we needed.

Second, many companies operate in the realm of requirements. This is prevalent in software development organizations, but also contracts between companies and agreements between departments in companies are typically specified in terms of requirements. In my experience, every requirement is described as a means to achieve a specific set of outcomes. Earlier, we didn't have the data to establish that the outcome was achieved and consequently, we used requirements. However, focusing on desired outcomes leads to a much healthier and much more data-driven way of operating and avoids the tendency of many to dictate solutions in areas where they have little to no domain knowledge.

Third, planning is a big topic in most companies. However, with fast feedback cycles through, for instance, DevOps, we can let go of planning as the coordination mechanism and adopt experimentation instead. The problem with planning is that it makes assumptions, many of which are simply incorrect. We tend to operate in a complex belief structure that's largely unfounded and disconnected from reality. By defining hypotheses and conducting experiments to confirm or disprove these hypotheses, we can develop an open-minded, learning-oriented organizational culture that's much better connected to reality.

Finally, even when experimenting, we tend to rely on humans to define the topics for experimentation. Where feasible, this should be complemented or even replaced with automated experimentation and optimization. Approaches such as reinforcement learning and multi-armed bandits allow for fully automated exploration of solution spaces and may result in solutions that no human would have ever conceived.

Data-driven ways of working are often agreed to in theory but seldomly followed in practice. This isn't ill-intended but rather a natural consequence of the way we're wired as humans. To address this, we need to put mechanisms in place to ensure that we use data-driven ways of working instead of opinions, beliefs and politics. These mechanisms include ensuring data availability, focusing on outcomes, experimentation and automated exploration of solution spaces. Einstein famously said that not everything that can be counted counts and not everything that counts can be counted. However, when it counts and it can be counted, we better do so!

ORGANIZATION DIMENSION

There are few topics in companies as sensitive as the notion of organization and organizational setup. Many still view their career as climbing the corporate ladder and, having reached a few rungs up the ladder, worry about sliding down again. So, every discussion that might rock the boat is received with great skepticism and people easily get defensive.

The fact of the matter is, though, that in times of rapid change, the traditional, hierarchical setup is probably the worst possible way of organizing large groups of people. Hierarchical structures are good at scaling, controlling many people from a central position, they're very efficient for repeatable tasks, allow for harmonization of processes and effectively support globalization. If the context in which the company operates is stable and has a low level of complexity, hierarchical setups are perhaps the best answer to the organizational challenge. They're extremely good at exploiting the learnings of the past and driving continuous efficiency improvements.

Of course, hierarchical organizations also have weaknesses. They tend to suffer from slow decision-making. Power tends to be driven by position and not capability. Because of the career ladder notion, the company tends to be internally focused and things easily gravitate toward politics. Inherent in the structure is the extremely high resistance to changes. Hierarchical organizations are, in many ways, even intended to naturally resist any change to the status quo. If the context in which the company operates is volatile, the future is unpredictable and the industry is undergoing a fundamental transformation, we need a much higher degree of exploration. In a hierarchical organizational culture, however, exploration is very inefficient as it has a very different reward function in that only a small percentage of the things we try out will actually work.

Guess what: the digital transformation is volatile and unpredictable. We don't necessarily know how things will unfold and, especially, when they will unfold. So, we need to operate with a clear strategy of how we want the industry to evolve and at the same time act with agility in response to the context in which we're operating.

This calls for a new way of thinking about organization. There are numerous views on this, including many by people who have spent much more time thinking about this question and are much smarter than me. However, in my view, there are at least three main aspects that need to be considered: cross-functional teams, new skills and different leadership.

Cross-functional teams are incentivized to reach certain quantitative outcomes and contain all the skills needed to achieve these outcomes. The notion is that intra-team coordination is at least one order of magnitude (if not more) efficient than inter-team coordination. So, by bringing individuals with various skills into the team, we're increasing the efficiency of communication dramatically.

Second, we need to train and hire for new skills. There are two types of skills to consider. First, to maximize the use of data and AI as well as continuous business models, we need people in the company who have the relevant knowledge and who can continuously develop and grow their understanding of the domain. The second type of skill is being comfortable in the unknown. This requires a relentless curiosity, a willingness to learn and to be surprised by what happens. To explore and embrace the perception of inefficiency that comes with it and to trust that the process will lead to the desired outcomes, even if it's not clear right now how we're going to get there.

A great example of this is A/B testing. When defining an A/B experiment, you need to start from the mindset that you don't know the answer to the question the experiment is hoping to elucidate. Also, the vast majority of A/B experiments in most SaaS companies fail in that the experiment doesn't result in a better outcome for the company. This requires the curiosity to use even the unsuccessful experiments as a basis for learning. Finally, you have to trust the process and accept that 95 of 100 experiments (or more) won't result in a positive outcome but that the few successful ones will justify the overall investment of time and resources.

Third, leadership. Traditional leadership tends to be intertwined with management and, especially high up in the organization, tends to be driven by short-term financial metrics. Leadership in digital transformation means letting go of preconceived notions based on earlier 'old-world' experiences and leading people even if you're uncertain and, most likely, the least knowledgeable in the team. Bringing people together around curiosity, learning and experimentation, leading them through the many unsuccessful outcomes and helping everyone view this as a learning opportunity is a different type of leadership. Especially as leadership will be much less based on positional power. In addition, digitalization often requires decisions that hurt from a short-term perspective but are instrumental for creating a desirable future. We need leaders who are willing to place these kinds of bets.

Hierarchical organizations are the least suitable form of organizing large groups of people during industry upheavals such as a digital transformation of your industry and company. Instead, we need to explore alternative ways of organization. Even if there are many aspects and views on this topic, in my view at least the notion of cross-functional teams, developing new skill sets, both technical and personal, as well as new forms of leadership need to be realized to be successful in this transformation. Remember: most people are well-intended so if change is slow, it's likely due to the way you've organized your people.

ORGANIZATION: CROSS-FUNCTIONAL TEAMS

When I was 14, I started to work in a local supermarket. That was actually not legal, but family connections and some creative managing of practicalities caused me to stomp around there anyway. The guy who got me introduced to the secrets of filling shelves, pricing goods and unloading trucks had the habit of not just telling me what to do but also how to do it. For the first few hours, that was OK, but it rapidly started to rub the wrong way. Although I thought I was unique, it didn't take long until I figured out that nobody likes to be told how to do their job.

The funny thing with traditional organizations is that telling people how to do their job is at the heart of how these companies work. This goes back all the way to Henry T. Ford, who managed to build a highly productive, scaling car factory largely using uneducated labor. The innovation was to slice the end-to-end work task into small pieces so that each individual could do their little part without needing to understand the whole. The idea was that the intelligence was deployed elsewhere.

Even if most modern companies would claim that they've left that kind of thinking behind, in practice quite a few of the basic principles stay the same in functional organizations. Even in R&D, the notion of mechanics, electronics and software doing their respective things with a governance layer of systems engineering still is the norm in many organizations.

For highly repeatable work, the separation into small slices that can be performed by an individual remains a very efficient way of accomplishing outcomes. With one caveat: we've become very good at automating anything repetitive. This means that if work is repetitive, it has almost certainly been taken out of the hands of humans and given to a machine of some kind.

Once you digitalize as a company, the work that isn't automated and remains in the hands of humans is work that's not repetitive but rather creative. And of course, it doesn't exist in isolation but needs to integrate into whatever already exists, requiring a holistic view. This is where cross-functional teams shine. A cross-functional team, either project-based or permanent, is asked to focus on achieving a specific outcome but isn't told how to arrive there. Instead, it has all the skills and capabilities inside the team to hypothesize, plan and execute to achieve the intended outcome.

In digitalized companies, everything that can be automated is automated to minimize human involvement in anything repetitive. This is of course great for humans as we hate repetitive activities but also good for the company as the risk of human error is reduced as well. It allows us to instrument and measure the automated processes so that we have a complete and quantitative overview of the performance of the business. This overview can be used to identify the improvement areas with the highest potential.

Cross-functional teams are then asked to “move the needle” on one of these high-priority areas (while avoiding significant negative effects elsewhere). Rather than telling the team how to do their job, we trust it to have the skills and capabilities to figure out by itself how to proceed. The evaluation of the team is then no longer a qualitative and often rather random sample of the way it’s working but rather a quantitative one as the team either moved the needle or it didn’t.

Some may feel that giving teams that much freedom causes a “free for all”, “let a thousand flowers bloom” kind of situation, but in practice, teams are focused on the highest-priority areas for the business and evaluated on accomplishing tangible, measurable results in their area of responsibility. The concept should be that the team owns its own outcome and is unable to hide behind excuses outside its control.

Digital companies automate everything repetitive and use humans for creative tasks. By instrumenting the business and measuring performance quantitatively, teams can be asked to move the needle on specific KPIs. To be successful, they need all the skills and capabilities to accomplish the intended outcome without relying on others, hence these teams are cross-functional. Due to the quantitative nature of goal setting, they’re highly accountable and incentivized to deliver on the indicated outcomes. As the saying goes, activity isn’t the same as progress. And many traditional companies reward activity, whereas digital companies tend to reward progress. What are you rewarding?

ORGANIZATION: NEW SKILLS

As an engineer, I've often had difficulty with the somewhat touchy-feely language used in leadership literature and HR departments. Rationally, I know that humans are irrational beings that excel at post-rationalizing their decisions and actions. And to reason about our irrational or pre-rational side, we need to use language that seeks to model that part of us. It's just that it seems so incredibly hard to operationalize and turn into action.

Having said that, over the years and while working with dozens of companies, I've started to develop a gut feeling about the key challenges they experience. And often, their rational, clear and explainable limitations are driven by deep patterns in their culture and the personality traits of their leaders.

In most organizations, the leadership consists of people who excelled in the technology that was important one or two decades ago. We see in automotive and other industrial companies that most senior leaders have a mechanical engineering background. Although these skills were incredibly valuable at the time and often the basis for the careers of these leaders, the education and experience come with a lens that makes people almost unconsciously reject alternative viewpoints.

During the transition to smartphones in the mobile phone industry, many senior leaders were unable to imagine the importance of apps and a rich ecosystem of apps for their user base (I worked for Nokia at some point in the past). Currently, I meet many senior leaders, especially in industrial companies, who find it difficult to imagine how data can be valuable and an asset that can be used and monetized.

After the fact, it's really easy to recognize what we should have done. As the saying goes, hindsight is 20/20. The hard thing is to recognize what's happening and act accordingly in the moment as the disruption is unfolding itself. And even if you can see how the future will unfold, the next challenge is to get the company to follow along. In many cases I've been part of, by the time the organization gets in motion around a change, it's so obvious to everyone what needs to happen that we're way too late to play offense and all we can do is play defense and hope we don't get disrupted.

Many moons ago, I worked with a manager at Philips who often said, "We can't get it done faster, so we have to start earlier." This is all the more true in a digital transformation: we need to start building new capabilities and skills long before it's obvious that we need them. For most companies, it means

bringing in data scientists and AI experts, but for a fair share, it also means hiring many more software experts as the company was stuck in an ‘atom-centric’ rather than a ‘bit-centric’ world view.

The reason for bringing in these skills earlier is that the ‘lens’ through which digital natives view the world is fundamentally different from those with a mechanical or electrical engineering background. One small example is the start of production for a new product. Traditionally, the view was that everything is done and over with once we start manufacturing. In a digital world, things are just starting when we have the product in the hands of customers as only then can we start the feedback loop with software and model updates to the field and data coming back from it.

The new viewpoint brought by these new digital natives will collide with the current company culture and it takes a long, long time for cultures to change. Initially, the new views are rejected, then grudgingly accepted on the edges of the business and slowly, over time, become a more and more integral part of the company. Hence, start early as it’s hard to accelerate the pace of change.

Digital transformation requires bringing new skills into the company. These skills are especially concerned with software, data and AI, but they bring with them a fundamentally different culture and lens on reality. Initially, the company culture tends to reject the ideas and it takes time to absorb the new skills and their implications. So, you better start early to make sure you’re playing offense, rather than defense with the fear of being disrupted. Remember, as Steven Case said, the pace of change and the risk of disruption create tremendous opportunities.

ORGANIZATION: LEADERSHIP

There are few topics with as much hyperbole around them as leadership. Everyone is supposed to be a leader, either informally or formally. At least you need to lead yourself based on whatever you want to get out of life, both professionally and personally.

My experience is that many so-called leaders are managers who are very good at keeping the ship on the course that was put in place long ago. By attempting to keep tomorrow as similar to today and yesterday as possible, we create a sense of stability that allows us to specialize and become even better at what we were already good at.

This may seem very critical, but there are systemic challenges in modern hierarchical companies concerning exercising leadership. Some of the key ones include incentives, obedient R&D and excessive process compliance.

First, the incentives in the companies I work with are squarely focused on short-term results. What matters is delivering on the next quarter, the next product upgrade, the next marketing campaign, and so on. Anything longer term tends to be assigned to strategy and business development functions that, although considered relevant, are viewed as outside the mainstream in the company. As a consequence, even if individuals see the clouds on the horizon and want to do the right thing, the entire organizational culture and incentives make it a very hard uphill battle.

The second challenge is that in most organizations, R&D views itself as servicing the business. In my posts, I've frequently mentioned the BAPO model that explicitly stresses that business strategy should be driving architecture and technology choices. However, while the business side can change its mind very rapidly, R&D typically takes quarters and years to operationalize a new business strategy in the product portfolio.

As a consequence, R&D needs to know the business strategy a few years out to prepare. In practice, no one on the business side is willing to commit to anything that far out. So, if R&D waits until business asks for something, it will be way too late. Instead, R&D needs to exercise leadership, predict or even define the business strategy on the time scale it needs and start executing. In reality, in most companies, it's R&D that sets the real business strategy for the company for precisely this reason.

The third challenge is process compliance and organizational roles. Any relevant innovation in digitalization will upset the apple cart quite fundamentally as it typically requires a change in business model and a transition toward continuous value delivery. Even if R&D can provide the offerings that facilitate this, we still need the rest of the organization to change their ways. In quite a few companies, salespeople are the most effective destroyers of relevant innovations as they're unwilling or unable to sell them either because of lack of incentives or lack of skills.

So, for all the hyperbole, we need leadership in digitalization. To me, leadership is convincing others to join you on a path that they might not have taken themselves. It requires painting a picture of what the future could look like as well as a picture of what bad things will happen to the company if we don't change. It requires outlining a path from here to there, recognition of the pain we'll experience but also of the short-term wins we can score along the way.

Leaders use many tactics. There's the David-versus-Goliath tactic, where a small group of digitalization advocates takes on the organization as well as the industry at large. The Chicken Little tactic, where we claim that the sky is falling, is also quite popular to instill fear.

For all the talk and hyperbole about leadership, most so-called leaders are managers who are rewarded for changing as little as possible. And even those who really want to drive change face an uphill battle driven by inverse incentives, a reactive R&D organization and existing processes and organizational roles. Still, we need leadership in digitalization as it's the only way to make organizations change. So, even if it may seem insurmountable, remember what Margaret Mead said: "Never doubt that a small group of thoughtful, committed, citizens can change the world. Indeed, it is the only thing that ever has."

USER INNOVATION DIMENSION

Innovation is one of those overloaded terms that in popular discussion can mean anything that you want it to. In some ways, it's similar to the parable of the four blind men and the elephant: depending on where you work on innovation, it will look and feel different.

As a consequence, I still need to meet the first company that will admit to not being innovative. All companies will have activities ongoing that they put under there, independent of what you think of it. So, I feel it's really helpful to put a bit of structure into the topic.

First, it's useful to distinguish between technology-driven and user-driven innovation. Both are important, but each requires very different approaches to the innovation process. In later posts, we'll look at technology-driven innovation, but here, I'm going to talk about user innovation.

User innovation is focused on identifying unmet or underserved customer needs that can be addressed better by the company. Of course, new technologies may often be the key enabler for this, but it can also be inertia in the business ecosystem, conservative customers or other causes of lack of innovation in a particular business domain.

One of the models I find particularly helpful is McKinsey's three horizons model. In this model, horizon 1 is about the company's mature businesses that today pay the bills. Horizon 2 is concerned with proven, but smaller and fast-growing businesses. Finally, horizon 3 is about finding new businesses that could become future horizon 1 businesses.

In my experience, all companies I work with are predominantly focused on horizon 1 innovation efforts, which tend to be more sustaining in nature. Typically, they're focused on improving some of the defined and appreciated KPIs of horizon 1 products, such as performance, efficiency and level of automation. So, when companies claim to be innovative, most often they're referring to the efforts in horizon 1. The good news is that these tend to be quite predictable in outcome, meaning that you have a decent understanding of the return of investment (RoI). In particular, the RoI tends to follow a gaussian bell curve and virtually all innovations have a positive return.

Horizon 3 innovation falls in the radical innovation category and requires a very different approach. We're looking to create new businesses that may be, and typically are, adjacent to our existing businesses but sufficiently separate that they can't be viewed as extensions. Horizon 3 innovation, consequently, tends to take more of a "let a thousand flowers bloom" approach, combined with a relentless, proactive and aggressive pruning activity to shut down ideas that don't show enough promise.

The typical models used in horizon 3 innovation tend to be along the lines of design thinking and lean startup approaches. The return function tends to be a power function, meaning that most of the ideas will fail but the few that succeed generate outsized returns that justify the investment in hundreds or thousands of ideas.

The source of ideas to evaluate tends to be some form of design thinking where the concept is to develop deep empathy with customers as a basis for brainstorming and idea generation. Once we have promising ideas, the next concept is the lean startup approach where you take ideas through a funnel to evaluate whether the problem is real, the solution suitable, the small-scale MVP successful and the large-scale MVP successful before scaling. I discussed this in a post a few years ago.

The third element of user innovation is interaction with real users. Especially for B2B companies, this is hard for two reasons. First, the people buying your product tend to be different from those using it. Second, the key contacts at the customer as well as in the company itself tend to be quite protective and look to avoid having others 'mess up' the relationship with the customer.

The challenge is that most companies operate based on a set of outdated, incorrect assumptions about the customer that potentially result in attempted innovations that fail to meet real customer needs and/or provide suboptimal innovations that don't fully meet customer demand. The only way to address this is through continuous customer and user interaction. For B2B companies, this means deeply understanding and having empathy for the needs of all stakeholders at the customer. This isn't achieved through surveys and customer interviews, but rather by shadowing customers, working with them, and so on.

All companies claim to be innovative, but in practice, most innovation efforts tend to be technology driven, focusing on horizon 1 offerings. To thrive in a digital transformation, companies need to adjust their approach to focus on more radical, disruptive innovations as digital technologies allow for fundamentally new ways of solving customer needs. This requires adopting a horizon model approach, employing design thinking and lean startup principles and engaging in continuous interaction with customers and users. Such an innovation approach will be experienced as inefficient by many, but industry best practice has shown that it's the most effective as it generates a small number of crazily successful innovations providing outsized returns that justify the approach. Innovation is hard, but slowly sinking into the moras of commoditization is much worse in the long run!

USER INNOVATION: HORIZONS MODEL

One of the most common questions I get when discussing digitalization of an industry and a company operating in that industry concerns resources. The typical line of argument goes along the lines of agreeing that it is relevant and important to explore digital solutions for the company, but that unfortunately no resources are available to be allocated to this. The current business or businesses are consuming all the resources in the company and we can't free up anyone to work on innovation.

The fundamental misconception is that it is reasonable for the current, dominant revenue generator to consume all the resources available in the company. Often the argument is used that this business is the one generating all the revenue and consequently it can and should consume all the resources as well. The advocates of this often forget that today's cash cow once was an early stage innovation that was kept alive despite the fact that revenue was highly limited, if not non-existent.

One model that I frequently present in these contexts is the McKinsey three horizons model. At one of my earlier employers, we used this model successfully and it helped the conversations around resourcing and prioritizations. The three horizons model organizes the businesses that a company is involved in, as the name suggests, in three horizons. Horizon 1 concerns all the existing, revenue driving offerings that typically are mature, experience limited growth but that generate the vast majority of revenue for the company. Horizon 2 businesses are smaller, fast growing and are hopefully future horizon 1 businesses if we can feed the growth long enough. Horizon 3 businesses are new innovative ideas that are being evaluated to determine if these have the potential to become viable growth businesses for the company.

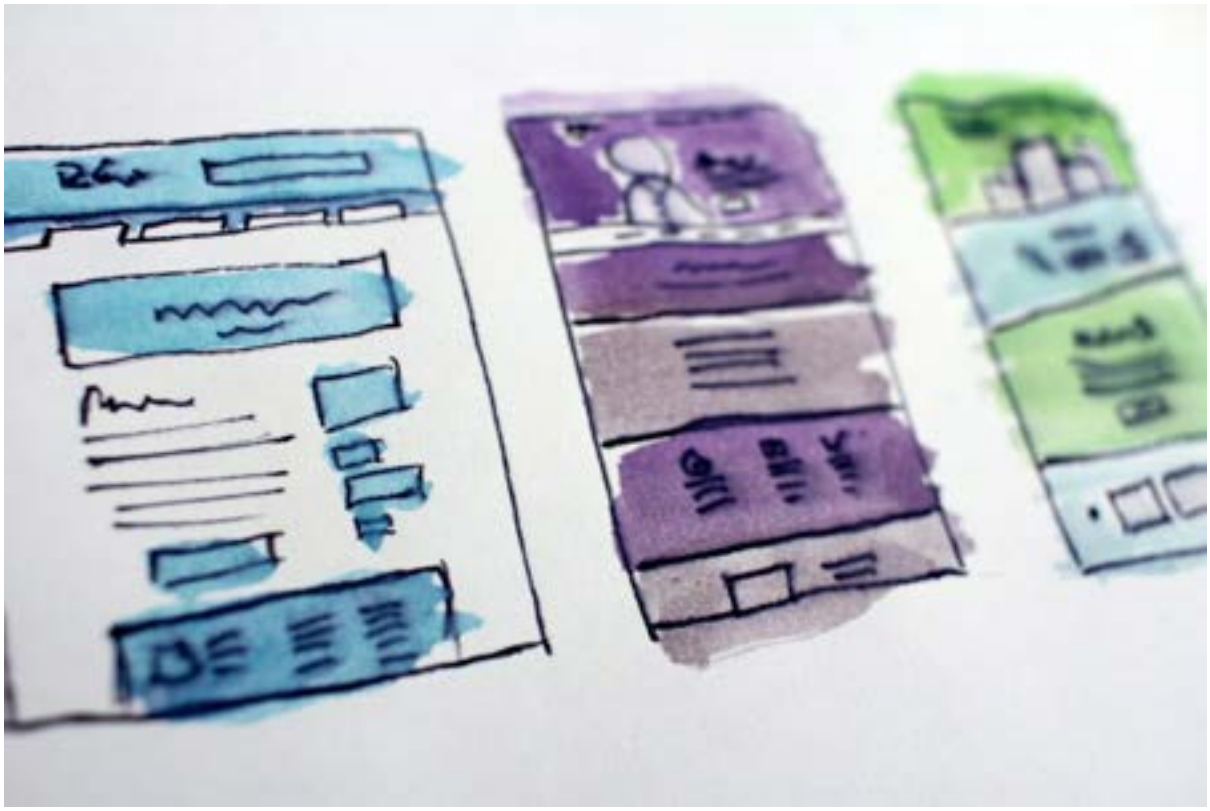
Each of these horizons has and should have different metrics for resource allocation and evaluation. In one of the companies that I worked for, horizon 1 received, in total, 70% of all resources in the company. These resources were divided over the various businesses that we had using their relative revenue as a metric. Interestingly, every year these resources were adjusted based on their growth percentage minus

10%. This meant that if your business was growing 5% the last year, your resources would be reduced with 5%. This often lead to significant complaints from the leadership for that business, but the line of reasoning is that if you have a business that is growing at the rate of GDP and consequently is flat, you need to focus on driving efficiency. Once a horizon 1 business reaches end of life and revenue starts to decline, the company needs to decide whether to spin out or sell the business, sunset it or to milk it as long as possible with minimal investment to squeeze all the revenue out of it before it disappears altogether

The second horizon received, in the case that I personally worked with, 20% of resources. Horizon 2 businesses are proven businesses that represent significant potential and that, typically, require outsized investment to capitalize on that potential. Hence, these businesses would receive resources beyond their growth percentage in order to accelerate growth, i.e. growth percentage + 10-20%. This means that if your business is growing at 30 or 40% per year, your available resources could easily increase with 50% per year. The key metric in horizon 2 is to drive and accelerate growth as much as possible and businesses that fail to drive growth and that are too small to become viable horizon 1 businesses are spun out or shut down.

Horizon 3 is a very different beast from the first two horizons as it is the birthplace of new innovations. Here the metric is not revenue or revenue growth, but rather the number of new ideas validated with customers. The third horizon receives the remaining 10% of resources, but the definition of what falls in horizon 3 versus the other horizons is often a source of debate. For instance, horizon 1 businesses investing in sustaining innovations for their offerings often considered this horizon 3, but that is not correct. Horizon 1 and 2 businesses need to fund their sustaining innovations out of their own budget and horizon 3 is concerned with uncovering unmet customer needs that offer the potential of completely new businesses. In subsequent posts, we dive into the mechanisms that we use for accomplishing this, but expect to see concepts such as design thinking and lean startup principles as well as a focus on customer interaction.

Concluding, many companies struggle with resource allocation because the main revenue driving businesses of the company have a tendency to consume all available resources over time. This is normal human behavior as there always is a crisis to address that requires additional help and we always seek to reduce risk by throwing more people at the problem. The role of senior leadership is to provide a counterweight and to ensure resource allocation that supports ambidexterity for the company, i.e. creating a future while ensuring the present. The three horizons model is a simple, easy to understand model that provides principles and guidance to ensure that sufficient resources are available to drive smaller, high potential businesses and to drive innovation to create new ones. In the end, even if most of us are concerned with the priorities of the present, the future is where we spend the rest of our lives.



USER INNOVATION: DESIGN THINKING & LEAN STARTUP

One of the things that never ceases to surprise me is the amount of misconception around the notion of innovation. Reams of paper have been filled with reflections on successful innovation and the people behind successful innovation typically receive enormous amounts of attention in their industry. Of course, what remains unsaid in these conversations is that there is an enormous amount of survivor bias going on in the innovation community. The majority of innovation experiments fail and the few that are successful receive outsized attention simply by virtue of surviving.

The challenge of innovation, in my view, is that it is concerned with breaking one or more of the norms, habits, customs or beliefs in an industry and ensuring that this coincides with an unidentified shift in customer preference. This means that for experts in an industry any innovation experiment is ridiculous and bound to fail. Which has led to my favorite definition of an expert: someone who will tell you why something can not be done.

To drive successful Horizon 3 innovation (see previous blog post for a description of the three horizons model), we need to suspend disbelief and simply try things out with customers. In my experience, this proves to be difficult in most companies for at least two reasons. First, this notion of involving customers in innovation experiments typically means to expose customers to very early stage prototypes that are far from the normal product quality levels of mature products. Many key leaders in companies are highly skeptical of this as they are concerned with the company's brand and reputation with those customers. In response, there is often a push to invest more into the R&D of the innovation experiment to bring it closer to the "normal" product maturity and quality.

The second reason is that radical innovation initiatives are, by their very nature, extremely inefficient in that the vast majority of innovations fail and only a few are successful. Especially in companies that pride themselves on efficiency, accepting that radical innovation requires many failures to get to a few successes is very countercultural. Combining that with the "rule-breaking" nature of innovation causes a situation where traditional leaders easily end up in a situation where they claim it is obvious that a particular innovation was not successful and that it should not even have been tried out to begin with. This then discredits the innovation initiatives and, over time, support for innovation starts to erode.

During the last decade or so, two concepts have become more prevalent to address this challenge, i.e. design thinking and lean startup. The notion of design thinking is, at its core, concerned with deep empathy with and understanding of the customer. By spending time with customers, shadowing them as they go through their day, understanding the broad context in which your offerings as well as those of competitors are used, deeply investigating the "job" that these offerings are employed for, etc. the result is that the ideas that the innovation efforts result in have a much higher likelihood of success. Even though I still have to meet the first company that admits to not knowing the customer, the fact is that in most contexts, the company knows its customers only through the narrow lens of how their offering is used by them.

The second concept is the lean startup. Whereas design thinking is concerned with generating higher quality ideas that have a better chance of being successful, the lean startup concept is focused on testing these ideas with customers against the lowest possible cost per experiment and with the notion of iterating as rapidly as possible through subsequent experiments. The insight behind the lean startup is that most innovation ideas fail and only a few are successful. As we don't know which ones will be successful, we need to weed out the unsuccessful ones as rapidly as possible and to iterate over potentially successful ones in order to learn as quickly and as cost effectively as possible what it is that constitutes the essence of success. In an earlier post, I described our innovation funnel approach that is quite helpful in this context.

Concluding, horizon 3 innovation needs to focus on customers and users and breaks the existing beliefs about them within an industry. Successful innovations capture an unknown shift in customer preference and then provide an offering around this insight. As most radical innovation attempts fail, we need to focus on two activities. First, use design thinking concepts to build deep empathy with the customer and user in order to improve the quality of the ideas generated as part of the innovation process. Second, apply lean startup thinking to ensure the lowest cost per innovation experiment as well as the shortest time per experiment with the intent of running as many experiments as possible per unit of time and resource. Finally, ensure that leaders understand the principles behind horizon 3 innovation in order to avoid the "it's obvious that" trap and the perceived inefficiency of innovation. And remember, as Simon Sinek said, innovation is not born from the dream; it is born from the struggle.

USER INNOVATION: CONTINUOUS CUSTOMER INTERACTION

There is a wonderful “not safe for work” quote that states that assumption is the mother of all eff-ups. Many of the things that went wrong in my life can, when doing root cause analysis, be directly blamed on me having the wrong assumptions about the situation at hand or reality at large. Of course, this is not just true for individuals, but for companies as well. Companies are successful because the people on the inside know something that the people on the outside do not. There is a set of beliefs and associated behaviors that come with every company and that are at the heart of their success. For instance, in automotive companies, the ability to design vehicles based on platforms and to create premium as well as low-end products from the same basis allows for cost effective scaling of production beyond a product-centric approach.

The challenge is, of course, that successful innovations are eagerly copied by competitors and the initial differentiation is commoditized. As a consequence, companies need to continuously innovate in order to maintain their differentiation. Innovation is a highly overloaded term in many ways, but in the end it is concerned with a new idea and an ability to monetize this new idea. At the heart of the ability to monetize is the desirability of the idea in the eyes of the people that would be benefiting from it. The only way to assess desirability is to actually present the offering to customers and users in order to gather feedback.

In my experience, providing access to users and customers for innovation teams in order to gather feedback on new concepts is challenging in most companies and especially so in B2B companies. The reason is that the current customers are the ones paying the bills and are viewed as the ones that we serve with our offerings and associated customer support. This means that everyone interacting with

customers because of the existing product portfolio is highly skeptical of providing access to these customers as it may upset the current relationship.

A second reason to limit access to customers is that we do not just want to show new concepts to them, but also measure their behavior as they use the implementation of the concept. And one likely result is that some, but too few, customers love the potential product, resulting in a too small addressable market. Especially if the positive customers are also key customers for legacy products, it becomes quite a challenge to shut things down. This is exacerbated in the cases where we have managed to make customers pay for the new offering as they will easily feel entitled to continued support.

The above easily leads to a situation where innovations are developed for way too long based on little or no feedback from customers. That causes the cost for each innovation experiment to become very high and as a consequence, we can not afford for innovations to fail as the sunk cost is too high. This exactly the opposite from where we need to be, i.e. testing as many ideas as possible against the lowest cost per experiment

So, the consequence of lack of access to customers is that within the company we all start to act based on our assumptions, rather than based on reality. Difficult as it is, we need mechanisms that allow innovators to gather true customer feedback throughout the process. These mechanisms need to cover at least four phases. First, we need to be able to be around customers and use ethnographic techniques to build empathy with and understanding of the customer. Many companies that I work with limit their customer interactions to the negotiations table during sales and customer support in case things don't work as they should. This may give suggestions for new product features, but will most certainly not lead to new insights that result in new breakthrough offerings.

Second, innovators need the ability to talk to customers to get verbal feedback on problem hypotheses that were identified as well as on solution approaches that they would like to explore. The challenge is of course that customers, as well as any other humans, have a tendency to exhibit socially desirable behavior. Telling someone that you think their idea is stupid and won't work is hard for people, so we need to be smart in how we ask the question, but the cheapest way to invalidate an innovation concept is to not build anything at all.

Third, we need to put (very) early stage prototypes in the hands of customers to measure if what they say they will do is really what they do in reality. There are numerous examples of products that failed despite the most promising verbal feedback from customers. The world of fast moving consumer goods (FMCG) is littered with failed products of this category. Hence, we can not rely only on what customers say, but we need mechanisms to measure how people behave.

Fourth, we need to be able to measure willingness to pay. There are many offerings that customers happily use if there are no associated costs, but that do not deliver sufficient value for customers to pay for it. One of my favorite definitions of innovation defines it as invention + monetization. If you are unable to monetize, you have not innovated. So, confirming that customers are not only using your offering but also are willing to pay for it is critical for successful innovation.

Concluding, many traditional companies limit access to customers for the majority of their staff. The consequence is that everyone in the company operates based on beliefs and assumptions about the customers rather than facts, which easily causes high investment in innovations that are flawed from the beginning. Instead, companies need continuous interaction with existing and potential customers for verbal feedback, observing behavior and measuring willingness to pay. It's not innovation if you can't get paid for it. Remember, assumption is the mother of all eff-ups! Make sure your assumptions are validated!

TECHNOLOGY INNOVATION DIMENSION

Many companies are good at generating sustaining innovations in their mature, revenue generating products. We know what KPIs customers care about and we focus our efforts on continues improvement of these KPIs. For example, because of regulation as well as customer preference, the fuel efficiency of vehicles as well as their reliability has steadily improved over the years.

The challenge with digitalization is that the sustaining innovations that were successfully convincing customers to keep buying become less effective and customers want to see different forms of delivering value. As an example, the car that I drive (a German brand) has lane keeping and adaptive cruise control as a feature and I use it quite a bit. However, the functionality is not very good and the car has a tendency to act weirdly in specific situations. I am OK with that. The thing that annoys me to no end, however, is that I know that the functionality will stay equally bad for the entire time that I will own the car. Unlike my computer, my phone and other equipment, it will not get better!

Similar to how Nokia (for a wide variety of reasons that I will not go into here) missed the transition in consumer preference from product variants to apps on top of a product with minimal variation, many companies are at risk to miss a fundamental switch in customer preference due to digital technologies. As always, your customers will not be able (nor is it their job) to tell you what they want, but they most certainly will recognize it when they see it and change their buying behavior. Of course, you can keep going on momentum, brand and customer relationships, but you need to adjust or risk disruption.

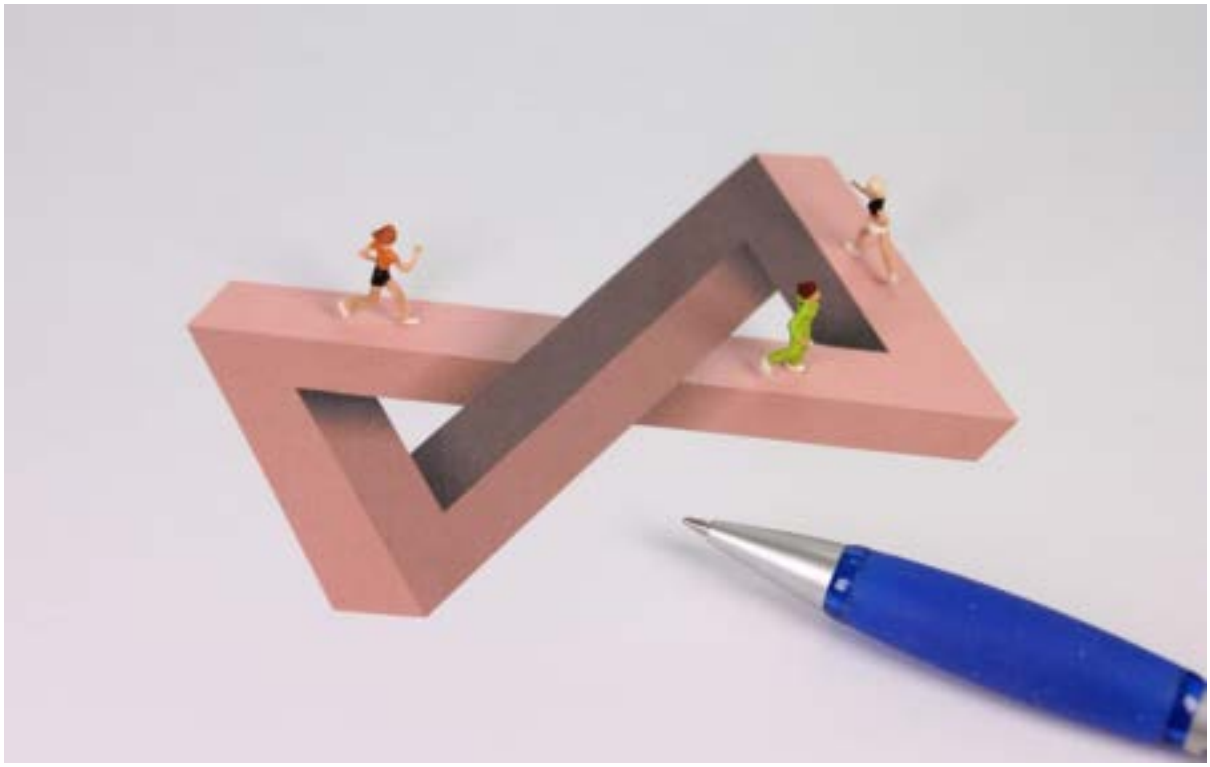
When it comes to digital technologies, there are at least three major technological approaches that need to be adopted, i.e. DevOps, A/B testing and artificial intelligence. The essence of digitalization is a fundamental shift in value delivery from transactional to continuous. For most companies, this can only be achieved cost effectively by changing the software in the offering, rather than anything physical. Frequent updating of software in deployed products of course brings us to DevOps. For digitally born SaaS companies, this is obvious beyond belief and industry best practice for close to two decades now. However, for many cyber physical systems companies, however, this is still a work in progress. There are many reasons, including regulations and certification as well as many of the company internal justifications that we have discussed in this series of posts to date, but the the fact that it is hard is no reason to not get there.

The second major technological approach is concerned with A/B testing and other experimental approaches. When we are able to deploy new software in systems in the field, we are also able to get data back from these systems. This opens up a quite significant shift in how we work with requirements and features as rather than guessing about the value to customers of new functionality, we can actually measure it. By deploying small slices of new functionality in some systems and comparing the key KPIs between systems that have the new functionality with those that do not, we can quantitatively and

statistically determine the impact of new functionality. That allows us to stop development of features that have no or even a negative impact and to double down on the things that really move the needle in a positive way. For anyone who has been in feature prioritization meetings between product management and R&D, the idea that we can decide what to include based on experimentation instead of rhetorics and storytelling, this should come as a relief!

No post on technology driven innovation can ignore artificial intelligence (AI) and this one is no exception. I have written about our work on AI and AI engineering in several earlier posts and my position has not changed: machine- and deep-learning (ML/DL) offers fabulous opportunities for new forms of value. In order to capitalize on that, though, ML/DL requires data, and often lots of it, in order to function well, which requires the constant flow of data from systems in the field. Similarly, ML/DL models should be subject to the same DevOps cycle (often referred to as AIOps or MLOps) as all other software in our systems.

Concluding, most companies are very good in technology driven innovation for their main revenue driving products. With digitalization, however, the innovations that drove product success in the past need to be complemented or replaced with digital technologies and technological approaches. Three of the main ones include DevOps to continuously deliver value to customers, A/B testing to quantitatively validate the value of new features and functionality before building it and artificial intelligence as it allows for much smarter system behavior in a variety of contexts. As Tim O'Reilly said: What new technology does is create new opportunities to do a job that customers want done.



TECHNOLOGY INNOVATION: DEVOPS

One of the models that I use frequently in my consulting assignments is the 10 types of innovation model by Doblin/Monitor Group. As the name suggests, the model identifies 10 different forms of innovation arranged around the business ecosystem, the offering and the customer experience. This model is important as most people think about improving the product when they hear the word innovation, but in practice, as the authors of the model show, investment in product innovation has the lowest return on investment (RoI) of all the 10 types. Instead, innovating the business model, customer experience and surrounding processes have significantly higher RoI.

One of the innovations that is not necessarily concerned with the product but with the other dimensions of innovation is DevOps. Although the term DevOps is used frequently, I still notice that many fail to identify how deep the impact of the adoption of DevOps is. It fundamentally changes, among others, the interface to the customer, the product in the field and, frequently, the business model. As we have discussed throughout this blogpost series, digitalization is concerned with continuous value delivery to customers which is predominantly driven by releasing of new software to products in the field. As the customer often is the owner of the product, there frequently is some involvement by the customer in order to ensure that the update happens when it is convenient for the customer.

The relation to the product in the field changes in at least two ways. First, traditionally software was extensively tested as part of the integration in the product before being shipped to the customer and this included several manual steps and activities. Now, the software is released and the product is expected to update itself without causing any issues or concerns. Also, the product needs to be able to roll back if an issue is detected in order to ensure availability.

Second, with the continuous connection to the product, we also want to collect performance and quality data and bring it back to the company for analysis. This connection forms the basis for all subsequent technology innovations, such as AB testing and the use of artificial intelligence.

Third, accepting a cost structure associated with continuous deployment of software to the field but failing to capture the additional value that you are providing is suboptimal in that it decreases margins. So, the adoption of DevOps often causes a change in the business model as well where the transactional business model is replaced or complemented with a continuous business model, e.g. subscription, usage or performance based.

In my experience, there are at least two challenges that many embedded systems industries are experiencing when digitalizing, i.e. certification and the operating models in the business ecosystem. First, most certification approaches assume a "one-off" certification of a system, including all its technologies. This certification activity is very labor intensive and expensive and consequently many companies are very careful to conduct a certification as infrequent as possible. This is diametrically opposed to DevOps as we want to update our systems in the field as often as possible. There are several approaches that address this and offer continuous models for certification, but in practice these have not yet seen major adoption in many industries yet. Depending on the class of certification needed, this requires process innovation not just at the company level, but at the business ecosystem level.

Second, the operating models in the business ecosystem often complicate the adoption of DevOps. For instance, in the defense industry, the customer, typically the military of a nation state, assumes a transactional model where all requirements for a new system are defined before a tender and the winner is selected based on price while satisfying all requirements. Again, this way of operating again severely complicates the adoption of DevOps and requires changes at the business ecosystem level before the full benefits of DevOps can be captured by everyone involved.

Concluding, although the term DevOps is used widely and many assume that it's simply the combination of development and operations, in practice DevOps significantly changes the relationship with the customer and the products in the field as well as the business model. DevOps is a keystone technology that drives many changes in the company as well as in the business ecosystem. The adoption of DevOps is often complicated by certification needs and existing work practices in the business ecosystem which requires a multi-pronged change management approach in order to realize it. But, as we all know, the best way to get better is through continuous improvement. As Mark Twain said, continuous improvement is better than delayed perfection!

TECHNOLOGY INNOVATION: A/B TESTING

Although it is uncertain who said this, a famous quote is “It ain’t what you don’t know that gets you into trouble. It’s what you know for sure that just ain’t so.” This notion is at the heart of digitalization: much of what we think we know is no longer true when the industry goes through a digital transformation.

Successful companies typically know something that the rest of the industry does not and this generates their success. This “secret sauce” may benefit the company for a long time, decades even, but when significant disruptions happen, all assumptions need to be revisited and reevaluated.

Most companies hold what we refer to as shadow beliefs. These are beliefs that may have been true in the past, but that are no longer true today. However, all in the company still operate based on the shadow beliefs and, in a sense, are focusing their energy on the wrong thing.

The only way out of this conundrum is to continuously test your assumptions with customers and the market and one of the most effective ways to do this when it comes to the product is by conducting experiments. When designed correctly, experiments provide statistically validated results that provide irrefutable evidence concerning the hypothesis that underlies the experiment.

In software intensive systems, experimentation is often conducted in the form of A/B tests. The basics of an A/B test are exactly what the name implies: we develop an A alternative and a B alternative for a particular feature or aspect of our offering. We deploy both the A and B alternatives and randomly assign users to the A and B groups. We measure the behavior of the A cohort and the B cohort and once we have enough data to derive a statistically validated conclusion, we deploy either the A or the B alternative to everyone and conclude the experiment.

When working with traditional companies on introducing A/B testing, one of the interesting topics that often comes up is that it is actually unclear to most in the company what they are optimizing for. For instance, in a workshop with a company in the automotive domain, we discussed adaptive cruise control (details are deliberately vague). However, when trying to quantify what constitutes a better adaptive cruise control feature, it became clear that the dozen or so people in the room did not have a clear view on nor alignment on what a successful adaptive cruise control looks like.

The consequence of vagueness on the desired outcomes is obvious: inefficiency. When different people have different views on success, they will each take decisions and act in a way that aligns with their own beliefs. And these decisions and actions may easily conflict with each other, potentially canceling each other out. Secondly, even if you align within the company on what constitutes success, it may still not be what the majority of customers would prefer.

Digitally born SaaS companies tend to run thousands upon thousands of A/B tests continuously and as a result are extremely data driven. This requires these companies to be very clear on the factors that they are optimizing for, often resulting in a hierarchical value model (see also this post).

As we'll discuss later in this series, we need the value model not just for A/B testing, but also for using AI models. Any model that is to be trained needs to know what is better and what is worse in order to optimize itself. You can do this through examples (labeled data sets), but also through quantitative models.

Concluding, traditional companies often suffer from shadow beliefs once they enter a digital transformation: the things that perhaps once were true are no longer true, but we act as if they were. In response, we need to become very precise and quantitative in terms of what we aim to optimize for and then validate any potential improvement using experimental techniques such as A/B testing. As the Cheshire cat told Alice in Wonderland, if you don't know where you want to end up, any road will do. And most of these roads do not lead to where you want to go.

TECHNOLOGY INNOVATION: ARTIFICIAL INTELLIGENCE

Every few weeks, it seems, we are treated to another major breakthrough innovation in artificial intelligence. Whether it is Deep Mind's system beating the world champion Go player, GPT-3 dazzling us with Turing test breaking abilities or DALLE-2 generating images that are simply stunning, it seems like we are on the cusp of general artificial intelligence.

In reality, my view is that we are much less advanced than what these news making innovations seem to suggest, and for the typical company that I work with, many of the machine learning and deep learning (ML/DL) models are still living mostly in prototypes and experimental setups.

For companies to be successful in their digital transformation, it is not sufficient to use more software and collect more data. We also need to use this capability to collect data for more than traditional data analytics. The data provides an excellent basis for training ML/DL models as well and if we do not exploit this opportunity, others will and out-innovate us.

For all the hype around artificial intelligence, I still see quite a bit of hesitation to go beyond the prototypes into production. This is the case for, at least, four main reasons. First, putting a significant part of your offering's functionality in the hands of a model that you don't really understand, can't inspect nor explain to customers is a hard thing to accept for anyone with a technical background. As a consequence, it is easy to decide to ask for more time, more evidence, more experimentation, etc. before you are willing to incorporate the model into a real offering.

Second, one of the key challenges of ML/DL models is their statistical nature. Whether you focus on accuracy, F1 scores or any other metric, the fact of the matter is that no model reaches 100%. This means that there will be cases, where the system simply acts incorrectly. Even if accuracy is 99.9%, it still means that, on average, once every 1000 instances, things go wrong. For companies that have thousands or millions of products out in the field, this leads to significant numbers of failures. Even if the model on average does much better than the algorithmically programmed solution, the uncertainty associated with the consequence of failures leads to a lack of deployment of these models.

One striking example for me is autonomous vehicles. The statistics show that these vehicles are much, much safer than human drivers, but of course still make mistakes with potentially lethal consequences.

The mistakes by these ML/DL models, however, are experienced as much worse as mistakes by human drivers. Despite the many videos of autonomous functions in cars avoiding accidents where their human driver missed the danger, humanity apparently rather has tens of thousands of people per year die in traffic accidents than accept a much lower number of deaths due to autonomous vehicles.

The third reason why companies are slow in deploying AI solutions is the associated work that comes with it. Most ML/DL solutions simply are data hungry and perform better with increasing amounts of data. Collecting, cleaning, labelling and storing vast amounts of data in continuous data streams is a major investment and very labor intensive. This goes counter the general public's view of throwing a ML/DL model at a bunch of Atari computer games and the system learning by itself. Most AI approaches are using supervised learning which typically requires the large data sets.

Fourth, some problems that may seem easy in concept prove to be surprisingly hard to get right in practice. Similar to the infamous Clippy by Microsoft in the 1990s, it may prove to be very challenging to provide solutions that provide the right predictions, classifications or recommendations and once system performance falls below a certain threshold, users are simply disgusted with the system and refuse to use the "intelligent" functions. Combined with the lack of AI expertise in most companies, many obvious use cases prove to simply be elusive with current knowledge, architecture and approaches.

Although the above reasons, as well as many others, may seem convincing to delay the adoption of AI, my point is the opposite: we do this because it is hard. The benefit of successfully deploy models that solve real problems that earlier were unsolved is significant and even if investing in AI is more risky than a run-of-the-mill innovation project, the fact is that AI represents an amazing general technology that affects all industries. That means, investing in continuous deployment, data pipelines to collect the data streams, etc. with the intent of also building the capabilities to successfully deploy ML/DL models over time.

Especially for embedded systems companies, however, there is one additional approach that often is ignored. Most AI these days is based on offline training using centralized data. However, embedded systems companies have thousands if not millions of systems out in the field that have, at different times, some excess computing capacity. This allows for federated learning approaches where you can have systems train their own models and exchange these with each other in order to continuously improve performance without having to bring all data to a central location. In addition, for less safety-critical use cases, we can use reinforcement learning to have systems experiment while in operation and share the findings with each other using federated approaches.

The holy grail, at least to me, is to have systems that are continuously improving their performance not just through continuous deployment of new software but also because these systems experiment with their own behavior and learn, over time, how to optimally operate in each specific situation and context. Once you have adopted A/B testing, the topic of the previous post, it is not such a large jump to identify areas of functionality where you allow the system to experiment autonomously.

Concluding, for all the hype around AI, many companies are slow in deploying ML/DL models in production. This may be because of the lack of explainability, the statistical nature of ML/DL models, the large amount of work setting up everything data and training related or the challenge of modeling the use case in such a way that we have acceptable performance. Still, AI is a general and transformational technology that offers many advantages if used well. We can not afford to not invest in it even if there are risks and likely several failures along the way. As Winston Churchill said, success is going from failure to failure without loss of enthusiasm.

AUTOMATION DIMENSION

As humans, we tend to avoid situations where we experience pain or discomfort. Consequently, many organizations tend to decrease the frequency at which 'hard' things are done. And rather than fixing the root causes of the pain and discomfort, often there's a desire to increase the amount of formalized process around the action that needs to be accomplished.

A typical case is the deployment of software. As it's a pain to test all new functionality in the latest version of the software and to regression test all the functionality that's supposed to be still working, we tend to decrease the release frequency to reduce the amount of release overhead. The argument I often hear is that customers are asking us to decrease software releases because it also causes them a lot of overhead to deploy the new versions. This argument is happily accepted by the company and used as an excuse.

Of course, it's orthogonal to the goal of digitalization, which is concerned with the continuous deployment of new functionality. Consequently, one of the tenets of Agile practices is that if it hurts, you should do it often. The idea is that if you often experience pain in a particular situation and aren't allowed to avoid it, you'll be inclined to start fixing the things that are causing the pain in the first place.

One of the key factors in people experiencing situations as painful is repetition and manual drudgery. As humans, we're terrible at conducting actions over and over again and even though a lot of manufacturing traditionally was about asking humans to conduct the same operation for entire work shifts, also in that part of industry, we've moved to teams jointly assembling entire systems from beginning to end.

The main answer to repetitive tasks is, obviously, automation. Computers are world class at conducting the same action over and over again without complaints, loss in quality or variance in output. So, it's

evident that when it comes to digitalization and continuous deployment of new software, automation has to play a major role. The three typical application areas focus on test, deployment and configuration.

In earlier posts, we discussed the importance of continuous integration and test. The idea is that every time an engineer checks in new code, immediately a new version of the software is built and tested. For most systems, the check-in frequency is much higher than the build and test cycle and multiple check-ins are combined, but the basic principle is to ensure that the software is always at production quality and that any defects are detected as soon as possible.

The second application area of automation is deployment. Especially in SaaS companies, it's custom to push every release of software that successfully passes the testing stage immediately into production. The idea is that if we build new features that are valuable to users, there's no point in delaying user access to them. For embedded-systems companies, this is much more difficult to accomplish as the systems receiving the software are out in the field. Traditionally, however, updating software in the field often was an elaborate process requiring a knowledgeable technician. Automation of the deployment process is to achieve a state where software can be updated without any human involvement and, preferably, without causing any downtime for the system being updated.

A third automation area is configuration. Different from SaaS, most software on systems deployed in the field has been configured and optimized by users to achieve optimal outcomes in their specific context. One reason why customers resist upgrading software is that it often requires them to repeat the configuration process. So, when preparing for continuous deployment, existing configurations need to be automatically incorporated by subsequent deployments. However, we can go one step further and have systems configure themselves without human involvement. This can range from rule-based configuration to reinforcement learning approaches where the system experiments with its own behavior to optimize performance.

There are at least three areas to consider when working on automation: finding the pains, modularizing processes and orchestrating. When humans are exposed to something for a long time, no matter whether it makes sense or not, we tend to start to believe that this is just how it is. Like gravity, there's no way to overcome it. The first step is to recognize that something that has been "just how it is" is actually fixable. The target of this activity is to identify activities conducted by humans that are repeatable in nature as these should be the focus of any automation effort.

The second area to consider is the modularization of processes. There's a 'dirty little secret' concerning automation: it tends to stifle change. The sunk cost of the process that has been automated is often significant and so is the cost of making changes. As a consequence, there's significant resistance against change in the broadest sense. This may easily result in suboptimal outcomes as processes aren't aligned with the current needs, but rather based on tradition and outdated beliefs. So, the focus of modularization is to break processes into relatively independent blocks that can be freely composed to decrease the cost of changing processes.

The third focus area is process orchestration. As processes are, where feasible, organized in these independent blocks, we need to compose the blocks within processes and the interdependencies between processes through orchestration. This includes both the triggering of process steps and the transfer of data involved in the processes. In recent years, quite a few new approaches and associated tooling around robotic process automation (RPA) have come to the market and these can be quite helpful.

Humans naturally tend to do difficult and painful things as infrequently as possible, which is counterproductive in a digital transformation. The Agile mindset provides a helpful tool in the notion of "if it hurts, do it often." That will cause organizations to streamline and automate processes and get humans out of the loop of repetitive actions. To achieve this, we need to identify the pains, modularize

processes to decrease the cost of change in the future and orchestrate processes. As Jon Gordon said: love the process and you'll love what the process produces.

AUTOMATION: FIND THE PAINS

As I walk around in numerous companies, one of the things that never ceases to amaze me is the number of people living out their lives in pretty poor jobs. Repetitive jobs, where they have lots of responsibility but little authority and where company processes, guidelines and rules dictate the way the work is to be done, with little to no freedom for the individual.

My typical response is something along the lines of "why the hell are you still in this job," though expressed more politely. The typical answers I get are that things aren't that bad, that there are good reasons why the job is structured as it is and that the individual has settled into the role. Of course, freedom is often defined as a cage where your wings don't touch the boundaries, so perhaps I'm overly sensitive to being constrained and straight-jacketed into a boring, repetitive job.

We've all seen the cartoon by KC Green where the dog sits drinking coffee in a burning house claiming that "this is fine" and, while being lit on fire, claims that things will be OK. And we've all heard the story about the frog staying put in a pot with heating water until it dies. These being cautionary tales, the fact of the matter is that many people settle and simply accept reality as it is, no matter how poor the circumstances are.

My main concern is that so many of these jobs could be automated to the point that computers would take care of the majority of the drudgery and leave us humans to the more creative and unique tasks for which we're primarily suited. Now, as a society, we've automated the last swaths of repeatable processes, so often the tasks not yet automated have some inherent difficulty that complicates automation. In my experience, there are at least three main drivers: inertia, regulation and forms of technical difficulty.

First, inertia is concerned with getting stuck in the past and continuing to do things a certain way because we've always done them that way. There's a part of us that appreciates and recognizes the

importance of traditions and easily abides by the rules dictated by these. This is difficult to change within the walls of companies, but even harder when these traditions cover an entire business ecosystem. A good example are shipping containers, which were invented as a concept before World War II, commercialized in the 1950s and still took decades to achieve broad adoption across the business ecosystem. This was despite the enormous reductions in transportation costs brought by these containers.

Second, regulation tends to freeze an industry in time based on what the best practice at the time of regulation happened to be. Although regulations are usually introduced with the best of intentions, in many cases they're a good reminder of why the road to hell is paved with them. Many regulations demand human effort in a process, leading to high costs and, consequently, slow processes as companies seek to reduce costs by minimizing the human effort spent on following the regulations. Any innovations that would replace humans, reduce costs and accelerate the industry are blocked by regulation. An illustrative example is the banking industry where fintech startups are trying really hard to disrupt way too profitable incumbents only to be shot down by regulators when they start to become even a little successful.

The third reason blocking automation tends to be technical difficulty. Some part of the process requires human intellect as we don't know how to automate it. This may include image recognition, interpreting video streams or processing natural language, to mention a few examples. The interesting thing is that AI has become incredibly good at these tasks and is more than likely to offer partial or even complete solutions.

The challenge is to be more like George Bernard Shaw's unreasonable man and to continue to adjust the world to oneself. Rather than accepting the status quo, proactively search for pains. Once we find these, refuse to accept arguments claiming that this was tried before and failed. Instead, approach your work and the work in your organization with a Zen beginner's mind, recognizing possibilities due to new technologies or other reasons and then acting on these. Don't act as a jaded expert who has seen it all, but aim to see every day afresh. In the end, innovation is seeing what everybody has seen and thinking what nobody has thought.

AUTOMATION: MODULARIZE PROCESSES

Automation is central for companies that are digitalizing, and new technologies, including natural language processing and image recognition as well as robotic process automation, allow us to automate processes that were impossible or prohibitively expensive to automate earlier. Although I'm a strong proponent of automating everything repetitive, there's a "dirty little secret" around automation that few talk about: once automated, companies often seek to avoid changing and evolving these processes as it involves potentially significant costs and risks. I see at least three key causes: deeply integrated implementations, lack of competence and interdependencies.

Especially consultants, but also IT staff employed at your company, often are extremely focused on the task at hand and finishing it. They largely or completely ignore the long-term consequences and implications. Project management tends to have the same priorities as any future changes to the resultant work product will be part of a new project and thus the problem of another project manager. Consequently, the implementation of processes that are automated tends to have a very short-term focus, resulting in implementations that do the job but are highly integrated and interconnected. This lack of modularization causes any changes to require significant effort.

A second challenge is that the people who did the original implementation often move on, either internally or externally, and consequently are no longer available when changes are required. As nobody really knows how the automated process works in practice, no one dares to touch the implementation as it may lead to unintended negative consequences. In software engineering, components that nobody dares to touch out of fear of breaking their functionality are referred to as "stinkers." Implementations of automated processes can easily end up in the same situation.

Third, in the implementation of data pipelines, we've seen several cases where people build undocumented dependencies on existing pipelines as they need some of the data flowing through them for their own purposes. In process automation, it's often quite easy to insert triggers for other automated

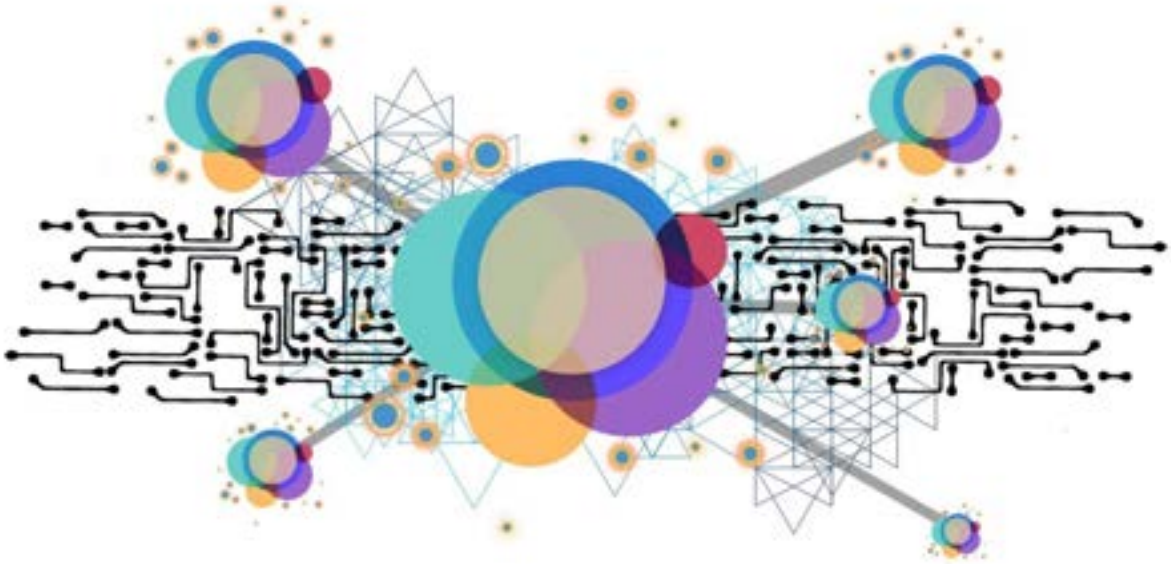
processes. If this is done in an implicit and undocumented fashion, it creates a spaghetti network of dependencies between different processes that complicates changing these processes as any change will most likely break something.

The reason companies should worry about hard-to-change processes is that these easily evolve into a competitive disadvantage. As the world moves on, better ways of doing things are developed that can't be capitalized on. Also, it becomes difficult to be innovative as many innovations require changes to at least some processes in the organization and these can't be evaluated as the associated cost is simply too high. So, the very thing that initially brought efficiencies and speed easily evolves into an anchor that slows everything down.

To address this, there are at least two main principles to observe: modularization and continuous investment. First, any design problem needs an architectural approach that defines the key principles. For anything in software, modularization is key. The notion of modularization is often poorly understood, but at its core, it comes down to decoupling units of functionality that are likely to change independently of each other and defining interfaces that allow one side to change without affecting the other side of the interface. Once automated, processes become software too and need to follow the same modularization principle.

Second, many IT-heavy companies tend to operate in a project-centric way. The products coming out of these projects are considered to be static and not requiring change. More than 50 years of software engineering research have clearly shown that the only software that doesn't need continuous change is software that nobody uses. Any piece of software that's in use needs a constant flow of investment to keep it up to date. This collides with the project-centric way of working and it's generally much better to take a product-centric approach where there are teams associated with each product. This maintains competence and ensures that the automated process is constantly updated to meet the current needs.

Although the automation of repetitive activities is critical for virtually all companies, there's a risk that years of investment in process automation result in an unwieldy, impossible-to-change IT landscape. This is a major competitive risk as it causes companies to resist changes due to the associated costs and risks and unduly drives up the cost of innovation efforts. To address this, companies need to embrace at least two principles: modularization and continuous investment. It's not, as Charles Darwin wrote, survival of the fittest, but rather survival of the fitting. Species that are unable to adapt fast enough to changing circumstances go extinct and the same is true for companies. Don't grow stale!



AUTOMATION: PROCESS ORCHESTRATION

There's little doubt in anyone's mind, I hope, that automation lies at the heart of all the progress in human well-being and economic development. It started with outsourcing physical labor to machines, initially through watermills and windmills and later through the use of internal combustion engines and electric motors. Later on, also "white collar" work was increasingly automated through the use of computers, and activities requiring intellectual labor were starting to get automated. This is a process that's still ongoing, affecting industry after industry.

There's an interesting pattern where an activity to be automated is first positioned and presented as supporting, with a human still running and in charge of it. The next step is to automate more of the work and to position the human increasingly as the supervisor of the automated process. Finally, with the increasing performance and reliability of the automated process, the human is removed from the loop entirely and the cost of the activity has dropped by orders of magnitude. This is why automation and the use of computers have such a strong deflationary effect, even if it may take a long time to capitalize on them.

One good example is the evolution toward autonomous cars. Currently, the majority of vehicles on the road have some sort of automated driver support with the driver still in control. The next step that's increasingly available in modern cars is a mode, especially on highways, where the vehicle controls speed, lane keeping and distance to other traffic participants, but the driver is the supervisor to ensure that the automated functions don't make mistakes. The final stage, which will be available in the coming years, is where the vehicle, at least in some operating domains, is fully autonomous and doesn't require human supervision.

Of course, many are concerned about this in terms of employment as, in the US alone, more than 3 million people work in transportation, and many of them will lose their job when autonomous vehicles can handle the majority of use cases. My prediction is that the first step will be for trucks to fully autonomously travel from a logistics hub outside one city via a highway network to a logistics hub close to another city. Human drivers can then manage the last mile into the city. Similar scenarios are developed for sea shipping. Even if this will affect many professions, I view this as the umptieth

disruption of work because of automation and I don't worry at all about us creating new professions for the people no longer required in logistics. During the 20th century, the percentage of people working in agriculture went from somewhere around half of the entire workforce to around 1.5 percent and nobody worried too much about the poor farmers getting out of a job.

For all the enormous benefits provided by automation, as discussed in the previous post, there's a "dirty little secret" concerning automation that few people talk about or reflect on: automation tends to stifle change and innovation. The cost invested in automating a process is often significant. Also, the knowledge of how the process was actually automated tends to be quickly lost as this is often done in a project and the project members disappear. Consequently, the cost of making changes is quite high and companies go out of their way to not change or to implement changes outside of the process. This may easily result in suboptimal outcomes as processes aren't aligned with the current needs, but rather based on tradition and outdated beliefs.

The best way to address large, complex problems very often is to break them down into smaller chunks and solve all the pieces individually before integrating them into a comprehensive solution. It's a basic, generic pattern applied to virtually all forms of problem-solving and design, and automation is no exception. The focus of modularization is to break processes into relatively independent blocks that can be freely composed to decrease the cost of changing processes. The question is of course what the right blocks are and how to compose them, ie process orchestration.

While it's difficult to provide generic guidance, we can identify a few patterns. Most processes are a sequence of activities. Each activity typically has a trigger, collects input data, performs an operation and often generates some output. In addition, there often are conversions of data, starting applications and federating coordination between different systems. Tools for robotic process automation often have a set of building blocks that can be viewed as generic components.

One aspect of process orchestration is the notion of interfaces as modules or blocks need to be able to interact with each other. An important development over the last decades has been to minimize dependencies between the sender and the receiver. Currently, the best way we have to accomplish this is to use a message bus to which all modules, blocks or components are connected. Messages can be sent and received but the sender is unaware of who receives the message and the receiver doesn't know where the message comes from. This allows for strong decoupling between different parts, which is exactly what we want.

However, there's still a need to orchestrate all these components, modules or blocks in such a way that the desired system behavior is accomplished. The challenge often is less concerned with describing the main, standard path in an orchestrated process, but rather to capture all the relevant process deviations, the interactions between different processes and the edge cases that can easily undo the benefits of process automation if human intervention is required frequently.

Even though automation has provided amazing benefits to humankind, there's a flip side few talk about: once automated, changing a process can easily become prohibitively expensive. The best way to avoid painting yourself into a corner is to modularize the processes you automate and facilitate the easy combination and recombination of the modules or blocks for process orchestration. Finding the right modularization and orchestration is often case specific, but failing to get this right may easily cause significant process debt that will be expensive to resolve. As William Edwards Deming said, "If you can't describe what you're doing as a process, you don't know what you're doing."

ECOSYSTEM DIMENSION

Western civilization is built on the notion of the sovereign individual: each person is unique with his or her rights, responsibilities, dreams and fears standing separate from others. Buddhist dogma takes the exact opposite perspective: the notion of self is an illusion and the goal of meditation and other religious practices is to free yourself from that illusion. Instead, we're all part of an inseparable consciousness and there's no real distinction between you and others.

Companies experience a similar challenge. Everyone is concerned with the company and, especially in larger organizations, the vast majority of mental energy is spent on internal issues. The fact of the matter is, however, that we all live and operate in social and business ecosystems and that the amount of interdependencies between us and others is often much more extensive than we realize.

In a way, this is a matter of perspective and focusing on the obvious versus the deeper reality. During the weekend, I read about Aspen trees. Each tree in a grove looks like an individual tree, but in reality, a grove originates from a single seedling and contains a root system that's much longer lived than all of the trees individually. The notion of an individual tree is actually false as all the trees as well as the root system are a single organism.

Of course, most companies do realize that they're part of a business ecosystem, but the understanding of the ecosystem is often limited. In an earlier post, I discussed the common traps. One of the predominant ones is the "descriptive vs prescriptive" trap, meaning that many companies consider their business ecosystem to be cast in stone and immutable. Those that understand that ecosystems can be changed tend to fall into the "doing it all at once" trap, the misconception that the only way an ecosystem can evolve is by changing everything and everyone in one fell swoop.

In my view, the better way to view ecosystems is as an equilibrium of forces. The interfaces between different players exist because the various forces in the ecosystem have resulted in a dominant design

adopted by all. However, by changing the forces in the ecosystem, it's possible to reach another equilibrium that's more advantageous for your organization.

Every ecosystem is organized around a platform of some sort that gives all the players a competitive advantage over those outside the ecosystem. The preferable role in the ecosystem, by far, is to be the keystone player who provides the platform. For any company of some reasonable size, the ambition should be to be that keystone player in your own space of functionality. In addition, positioning yourself as the preferred partner in the larger ecosystem where you're unable to be the keystone player is key. And, finally, the idea is to pull as much differentiation toward your own ecosystem at the expense of the larger ecosystem. The more you can change the balance of differentiation versus commodity in favor of yourself, the better off you are in the long run.

As we discussed in the architecture dimension, one important change that companies implement is the adoption of a superset platform. Initially, this superset platform is used to harmonize and align all R&D inside the company, but it of course provides a basis for building and strengthening a business ecosystem around your organization. This requires three main activities: platformize, exploit network effects and create multi-sided markets.

The first step is to platformize, meaning that you provide a common interface for your entire product portfolio such that third parties and customers can extend your offerings in a controlled fashion. The challenge is to ensure that you maintain a control point in the ecosystem that you're looking to create and expand as other companies are doing what they can to differentiate themselves and turn your platform into a commodity.

Once the ecosystem platform is established, the next step is to identify network effects within the ecosystem. Digital markets tend to have a winner-takes-all nature. Becoming the winner, however, calls for carefully designing and exploiting these network effects. Often this requires first building up the volume of participants for one stakeholder group and then, over time, bringing in other stakeholders that serve that first group. The ambition is to reach the 'ignition point' where new entities join the ecosystem of their own volition without you having to cajole them into it.

Once the volume of participants is significantly large and you have access to significant amounts of data generated by them, the next step is exploring opportunities to bring new stakeholder groups into the ecosystem by offering them the data from your primary participants in, typically, processed and aggregated form. When you manage to grow the ecosystem beyond the original stakeholders and exploit the network effects, truly multi-sided markets can be created that dramatically increase the ecosystem's power and value.

We tend to think of ourselves and our companies as individuals separate from the rest of the world. And even if we realize that there's a business ecosystem in which we operate, we tend to view it as immutable, at least by us. Instead, we should view the ecosystem as an equilibrium of forces that we can influence. Digitalization often results in a significant platformization effort within the company to support DevOps. This platform provides a great basis for creating or changing the ecosystem around us by offering an interface to others, exploiting network effects and building multi-sided markets with stakeholders that originally were outside of our ecosystem. Ignore your ecosystem at your own peril! To paraphrase Alexander von Humboldt: the most dangerous worldview is the worldview of those who do not understand their ecosystem.

ECOSYSTEM: PLATFORMIZE

Many of the companies I work with aspire to build an ecosystem around their product portfolio. The idea of having others complement your offering and increasing its stickiness while taking a cut of the revenue generated by these complementors is, of course, incredibly appealing.

As with most things in life, if something sounds too good to be true, it often is. To be successful in building an ecosystem around your portfolio, several aspects need to be in place. One of these is that the product portfolio has to be platformized around a common set of APIs and a common control point that ensures that the company maintains control of its ecosystem and doesn't get disrupted.

In my experience, product-centric companies often have difficulty adopting a platform strategy as the product teams need to give up a significant amount of freedom in how they work with their customer base. The interfaces provided to the customers of the various products need to be harmonized to ensure that third-party complementors can address as large a customer population as possible. These interfaces aren't only some APIs but also UI frameworks, standard workflows and standardized services, like authentication and data storage. Such harmonization across the product portfolio tends to lead to significant tensions.

Companies can be classified as project centric, product centric or platform centric. Each category has its specific processes, culture, norms and values. For example, a project-centric company, as the name suggests, maps all efforts to projects. Software-intensive systems that require a constant flow of maintenance, changes and new functionality often have a hard time in these organizations as the culture is organized around executing a project, finalizing it and being done with it.

Similarly, product-centric companies have great difficulty funding common infrastructure investments, such as basic platform functionality, as the funding, the priorities and the ways of working are centered around products. It's therefore extremely difficult to get a platform-centric way of working in place as it goes against the way these companies function.

Product-centric companies typically want to build an ecosystem around each product. In the vast majority of cases, this is a complete fallacy as the customer base for individual products is too small to

reach the 'ignition point' where the number of complementors and the number of customers using these complementors' extensions is large enough to be self-sustaining. We need the customer base for the entire product portfolio as a basis for the ecosystem to reach the ignition point sooner and with less investment.

Consequently, to be successful in building an ecosystem around the product portfolio, the company should transition from a product-centric to a platform-centric operating model. One of the key enablers is the superset platform approach. This requires organizing all of R&D around a single, common code base, harmonization of feature requests around a common governance mechanism, automated product derivation for all products in the portfolio, automated testing and, over time, deployment and DevOps. A superset platform approach allows for common APIs, UI frameworks and standardized services in ways that are much more difficult to achieve in a product-centric approach.

A key factor in opening up the product portfolio to complementors and customers is to ensure a control point, ie a mechanism that prevents others from disintermediating your platform and engaging in a direct relationship without involving you and your platform. As discussed in an earlier post, complementors will seek to push your platform into commoditization and become the sole providers of differentiation that customers care about. Once they're successful, the next step will be to separate from the platform altogether. The purpose of the control point is to ensure that this can't happen. Various mechanisms can be employed including forcing all complementors to use your authentication mechanism, storing all data on your servers so that complementors don't have their own data access and legal mechanisms that allow you to cut them off.

Although all companies aspire to be the king of their own hill, most are (also, mostly or exclusively) operating as complementors in another company's ecosystem. Building and growing a valuable business in that context also requires a proper understanding of that business' ecosystem as the platform provider can decide to move into your space and disrupt you by providing as part of the platform the functionality with which you used to make a living. Understanding the dynamics of the ecosystem can allow you to proactively move and take action before you become a victim of forces that are impossible to control as a complementor.

Every company desires to be a platform company but few understand what this entails in terms of preconditions, changes and practicalities. Especially reaching the ignition point in an ecosystem requires significant investment, a well-defined strategy and exceptional execution of that strategy. The payoff is amazing, but getting there is a hard journey. As former Nokia CEO Stephen Elop said, it's no longer a battle of devices; it's a war of ecosystems. And that's increasingly the case for many industries.

ECOSYSTEM: NETWORK EFFECTS

There's a beautiful saying in American English: build it and they will come. The idea is that simply building something and making it available will cause customers to beat a path to your door. To throw their money at you simply to get access to the amazing thing you created.

Of course, we know that this doesn't work in the real world. Products and services need to be marketed and actively sold. Building and manufacturing products is half of the job. The other half is selling it, getting it into operations and providing customer support.

When it comes to building an ecosystem around a product portfolio, I meet equally naive views. The general thought is that simply providing an API to the products will cause numerous complementors to flock to our ecosystem. However, even getting the approval and support within the organization to open up an API for third-party developers not seldomly is a humongous battle. Many companies are wired to keep things confidential and internal and seek to minimize information leakage to the outside.

Also, the decision to open up the product portfolio to the ecosystem is often triggered by customers and partners asking for it. Sometimes others have been asking for it for years and there may even have been, frequently successful, attempts to hack your products to extend them with additional functionality. This tends to reinforce the conviction among the proponents that "they" will come if we just build it.

Despite the strong market signals, building an ecosystem tends to be an uphill battle. It's very difficult to get to the ignition point where complementors and customers have jointly created a market of sufficient size to make it self-sustaining. However, when we succeed, we ride the typical pattern in digital markets: the winner takes all.

The key principle in accomplishing this is to focus on network effects. A network effect was first discussed with the introduction of the telephone. If you're the only person in the world with a telephone,

the value of the thing is zero. If there's one other person with a phone, there's some minor value as you can now talk to that one person. For every additional person acquiring a phone, the total value of the network goes up exponentially.

When building an ecosystem, the idea is to find ways in which you engage the typically very small number of current participants in bringing in new ones. There of course is a direct benefit for ecosystem participants, but typically this benefit is too small for anyone to take action. So, as the keystone player in the ecosystem, we need to offer additional incentives to activate the existing flock.

These incentives can be monetary, but they don't have to be. Especially in B2C markets, reputational incentives like certifications and other forms of recognition can be equally successful. Another potentially effective mechanism is gamification as many people enjoy engaging in friendly competition, also outside of computer games.

Especially in the case of monetary incentives, one aspect to consider is system abuse. Ecosystem participants care about the incentives but not so much about the goal you're looking to accomplish with them. Consequently, any way that allows them to game the system and maximize their incentives with minimal effort will be happily exploited. This can become quite costly for the company without having the effects to show for it.

The abuse may cause you to want to stop the entire incentive system, but that would be throwing out the baby with the bathwater. The challenge is to develop an effective system to exploit network effects at the lowest cost, taking into account that this easily leads to a cat-and-mouse game with the players seeking to abuse the system. Constant evolution of the incentive system in response to the learnings takes real work, but the reward will be immense if you're successful.

Successful ecosystems have reached a state where the network effects in the ecosystem are self-sustaining and require minimal effort from the keystone player. Getting there, however, calls for active engagement in building the ecosystem. The most effective mechanism is to identify, reinforce and introduce network effects through carefully designed incentive systems. You may wonder why you should pay to persuade others to join your ecosystem, but the fact is that most ecosystems are only sustainable with a significant number of participants and achieving that requires investment.

CONCLUDING THOUGHTS

Having worked on the digitalization topic for the last decade or more, with dozens of companies and in a variety of capacities, ranging from researcher to consultant, I keep being amazed at the multi-dimensional, highly interconnected nature of the challenge as well as the lack of understanding and the confusion around the topic. Of course, the purpose of this series is to provide more structure and understanding, but a blog can only bring people so far. Reflecting on the last months as I worked my way through the topic, there are a few concluding thoughts I want to share before moving on to the next series (on product development).

The first reflection is that the notion of continuous value delivery to customers is still highly controversial in many companies. My view is that the essence of digitalization is a fundamental shift from transactional to continuous value delivery. How this materializes is a topic for discussion in companies. Some discuss it only for services surrounding their product. Others consider it for the software in their products but not the hardware. The most enlightened ones are open to offering the entire product, including the 'atoms,' in a continuous value delivery model.

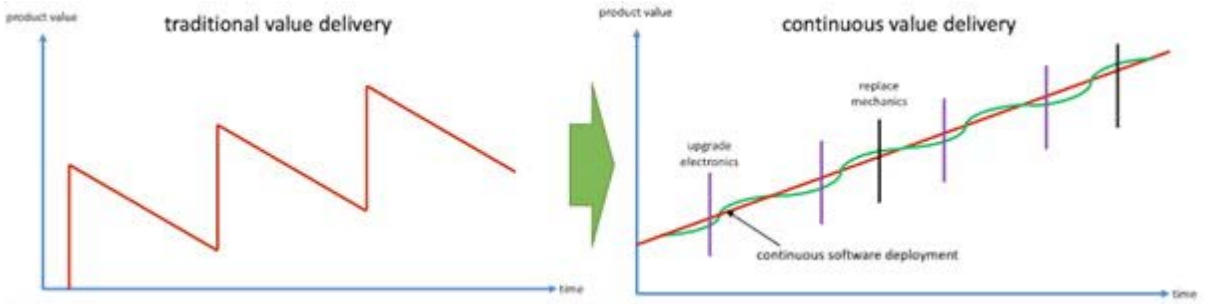


Figure: from traditional to continuous value delivery

A second reflection is that many companies still confuse continuous value delivery where the offering is improving over time with the delivery of a service and an associated service fee. There's nothing wrong with a service business model, but some companies try to sell their product through a transaction and then offer a service on top of it without providing additional value over time. Most customers refuse to pay recurring fees unless they feel that there actually is some value they're paying for. Why pay monthly for something that's just static and doesn't improve?

Third, it never fails to surprise me how poor an understanding most companies have of what it is that customers consider to be valuable and what they consider commodity, especially when trying to quantify this. The notion of continuous value delivery drives a conversation on what it is that actually and quantitatively provides value to customers. Here, many fall into the trap of the worthwhile many (rather than the vital few), meaning that there's a long, long list of important factors to placate everyone

but without a clear prioritization. Getting to the ‘vital few,’ meaning choosing a small number of the most important factors to focus on and ignoring everything else, requires leadership and is missing in most companies I work with. If you don’t know what you’re aiming for, how do you know you’re succeeding?

Fourth, successfully performing a digital transformation requires one to recognize its complex nature: everything is connected to everything else. The way we architect our products is an embodiment of the business strategy and a new, continuous way of delivering value to customers fails unless sales decides to actually sell it. Companies are complex, intricate and highly interconnected organisms and, consequently, finding the slices where we can drive progress in relative isolation from the rest of the company is surprisingly difficult. My experience is that a cross-functional team addressing a small, end-to-end slice is often the best way to create the first proof points. Once we’ve reached that point, scaling the approach still requires changing everything and everyone, but the proof points build confidence that we’re moving in the right direction.

Finally, the power of culture, including commonly held beliefs and underlying assumptions, never ceases to amaze me. I almost constantly have to use the “five why’s” mechanism when working with companies to identify the underlying reasons for the behavior I’m seeing. Almost always, what’s initially viewed by me and internal change leaders as the problem proves to be a symptom of a deeper cultural pattern or shadow belief. And frequently, we have to peel back multiple layers of the union to get to the true root cause. This is surprisingly difficult as virtually everyone in the company is blind to their own culture.

Digital transformation is hard. Not only because it’s so multi-dimensional and highly interconnected, but also because things aren’t black and white. It’s perfectly feasible to build and sell offerings using the old ways. The problem is that it’s an inferior way of doing things that, industry after industry, is rapidly being replaced by a better way embodied in the digitalization. People fear letting go of the known and comfortable as you know what you lose but don’t know what you’ll gain until you’ve reached the other end of the transformation. However, how long can we persist in an inferior way of operating in the market? To quote Hemingway when asked how a successful writer like himself went bankrupt: first slowly, then rapidly!