What's cooking in the industry?

Trends in software development

addalot

What drives development?

We are living in exciting times for software development. Software is becoming an increasingly important part of all businesses. Applications such as autonomous driving, AI and virtual reality have become part of our reality. Only a few years ago we regarded them as science fiction. New entrants wait just around the corner with innovative software solutions that will further disrupt existing markets. We can expect more from new technology areas like Cloud, IoT and Big Data. No doubt all organizations are challenged by this fast-moving, global marketplace. In this context, how is the Swedish industry facing up to the challenges? We wanted in particular to know:

What challenges are the most important?

What software methodologies and concepts are the most promising?

Interviews with some 20 development managers in different domains gave us a better understanding of what's cooking in 2018. Starting with business drivers we found that all companies in one way or another have the following three challenges on their horizon:

Increased competition Fast moving markets New technologies

- **The competition** requires higher productivity and faster innovation. The use of Open Source is is clearly increasing, as is the focus on services complementing or replacing products (Servitization). Many are also getting better at Data Driven Development by using the vast amount of data that they collect from their systems.
- **The market** is pushing for being adaptable. Agile development and Continuous Delivery are gaining ground in most domains, even in safety critical systems. Data Driven Development also plays a key role in adapting to specific markets.The market is also driving companies to provide a modern development environment to recruit and retain the best developers.
- **New technologies** are often heavily dominated by Open Source because they depend on substantial, existing infrastructure. As a result, Secure Development is becoming more important, as previously closed systems are opened up.

Following up on these challanges by also asking our development managers how they choose to tackle them, we found that it all boiled down to six promising methodologies or concepts.

In this report we go into more detail about the concepts and how our interviewed companies experience them. The solutions we outline are based on their needs. The bottom line is that we must know the business challenges to select solutions that best address the needs for change. Therefore, before continuing, we pose the following questions:

Which are your challanges? How do they impact your ways of working?

"If you do what you've always done you'll get what you've always gotten"

— Jessie Potter

Adapted Agile

All for Agile, Agile for all

From its humble beginnings, Agile has conquered the world with its promises of flexibility, reduced time to market and customer centric development.

Today there is broad agreement on the benefits of becoming more agile. But off-the-shelf Agile is not the solution for more complex environments. Fortunately, there are means to adapt Agile methods to the specific needs of almost any situation.

Continuous Delivery

Small streams make a big river

Releasing software to users is often a lengthy, painful and risky process. Paradoxically, the solution to this problem came from doing more releases – but in smaller chunks.

Continuous Delivery addresses the problem by establishing a highly automated delivery pipeline from code commits to market deployment. Small pieces, quick in, quick out or bounce back.

Industrial Open Source

Open up to Open Source

Facing a complex market where systems are no longer built exclusively internally, the industry has started to mix in-house, third party and open source software. Why, one might ask?

Open Source is the key to increasing development speed and lowering cost while boosting innovation. It's as easy as that. So, the next obvious question is: how? Well, this is exactly where Industrial Open Source comes in.

Secure Development

Better secure than sorry

With an increasingly digitalized society, cybersecurity has become extremely important. We must be able to trust that applications we use and data we store are not tampered with.

Secure Development addresses the need to identify and fix vulnerabilities during development, rather than after. It requires implementation of new security practices, but also leads to other benefits like bettter overall quality.

Servitization

From products to services

Numerous companies are currently complementing or even replacing their products with services. The rewards are many in terms of better competitiveness and profitability, and an offering that differentiates companies from their competitors .

In a transformation journey of this magnitude it's easy to get lost. Thus, there is a need for a servitization compass to guide you on the way forward.

Data Driven Development

"The customer is always right"

New technologies have enabled companies to collect an ever-increasing amount of data. Often in vain. But used correctly, the data can give a bettter understanding of the market and of customers..

Data Driven Development tells you how to unleash the power of data. Instead of (only) relying on experts, product development is directed by a data feedback loop with the customers.

Adapted Agile

Agile development is used in many different environments, although the level of implementation differs. Clearly it is here here to stay, and not without reason:

- Time for product definition is limited and it needs to happen in parallel with development.
- Development is becoming more complex and difficult to drive top down, favoring exploratory development by “expert teams”.
- Software companies must deliver new functionality fast in order to meet changing market needs.
- Customers expect small continuous updates rather than large chunks that impact operation.

Agile started in small IT settings where it quickly proved its value, but as with any solution to a complex problem, not even Agile is a silver bullet. Implementing Agile according to its original definition in other domains turned out to be difficult. This is where Adapted Agile comes in. There are several circumstances in which an organization cannot be fully agile, but needs to adjust its agile way of working:

- **Large-scale (complex) development:** Going from one co-located team to multiple teams spread over different sites will inevitably challenge the Agile concept. There are several models that address scaling (e.g. SAFe and LeSS), but generally there is a need to balance team independence with coordination and overall guidance outside of the teams as well as more focus on the architecture.
- **Supplier set-ups:** Organizations with several suppliers working in parallel on a complex system (as in the automotive industry) need more management of requirements, documentation and top-down decision making. An adaptedagile approach in this scenario could for instance be to agree on working according to an iterative agile process with less up-front requirements. The agile way of working will impact the commercial interface to the supplier and must be understood by all involved parties.
- **Hardware development:** Hardware development generally follows a waterfall model with long lead-times where latefound problems lead to increased cost and delays. The related software development cannot only prioritize agile concepts like customer value and learning from mistakes, but must support hardware evaluation and test. Some system testing also usually depends on the final hardware, which is not complete until late. Adapting Agile in this case could be done by decoupling hardware independent components of the software and working with simulated or replacement environments, so at least parts of the system can move towards Agile. This will require additional hardware and software planning and coordination, which is not bad in itself.
- **Regulated products:** Organizations working with safety critical products must adhere to the ISO safety standards such as 26262 for automotive and 62304 for medical equipment. There are also coming similar standards for security. These standards enforce documentation, traceability and specific activities such as hazard analysis. However, all these standards are written from a waterfall perspective. There are ongoing efforts in the community to adapt these standards to agile development. These adaptations divide the required activities into three phases: What needs to be done initially, such as hazard analysis? What can be done as part of each iteration? And what must be done after the last iteration? An example of a divided activity is security penetration testing that could be set up automatically during iteration, whereas more extensive penetration testing is done towards the end.

The key take-away is that in all these cases you can still aspire to be more agile. The fact that you cannot go fully Agile doesn’t mean that you have to throw in the towel. It is important to understand the conditions in your specific environment and given this, define the best possible agile way of working. It is all about going back to the agile principles and finding a balance. What is most important to your organization, working software or comprehensive documentation? As indicated above there are some approaches available, but don’t forget that apart from the methodology there is also the challenge of adapting the organization and its roles for an agile environment. In particular, organizations that have strong specialized roles can face a challenge.

Judging from the answers to the survey, agile methods have been tried by a vast majority. In addition, 70 % of the interviewed managers say that they need to adapt agile to their specific domain requirements, and 55 % have problems doing so.

Industrial Open Source

The Open Source movement is several decades old, but it wasn't until the turn of the millennium that major companies entered the game and increased the concept's momentum. Since then, Open Source has transformed the industry through Open Source based development. Some of the reasons for why the movement is rapidly growing include:

- Global development awareness and access.
- Maturing industries, standards and practices.
- Increased demand for reduced lead-times and development cost.
- Merge of domains, for example connectivity technology with previously closed applications.
- The snowball effect. The big players are doing it, so now everyone else wants to get involved.

There are many benefits from turning to Open Source:

- **Innovation** – Overall community capacity and diversity imply increased innovation.
- **Speed** – The fact that Open Source software offers de facto standard solutions to many problems reduces the time-to-market for product offerings.
- **Cost** – Open Source is free to use but comes with a governance of license compliance cost. Development and maintenance costs are shared by the community.

Consequently, today's industry faces a complex environment where systems are no longer primarily built on in-house development, but on a mix of in-house, third party and open source. SW companies are moving from development to integration.

Managing this complexity requires practices to tackle legal and security challenges. The open source licenses include obligations, which if ignored can result in copyright infringement, extensive rework and delays. The open source code is also open to all, leading to increased security vulnerabilities.

There are three maturity levels of Industrial Open Source management to consider when the decision to engage has been taken:

1 Intake, which governs how to use Open Source in a repeatable manner:

- Policies and procedures. To ensure a common direction within your organization, policies and detailed procedures for Open Source intake, compliance and contribution are essential.
- Organization and Roles. Responsibilities related to Open Source need to be assigned. This can be done by e.g. appointing an Open Source Officer and establishing an Open Source Board. In addition, ensure sufficient legal involvement.

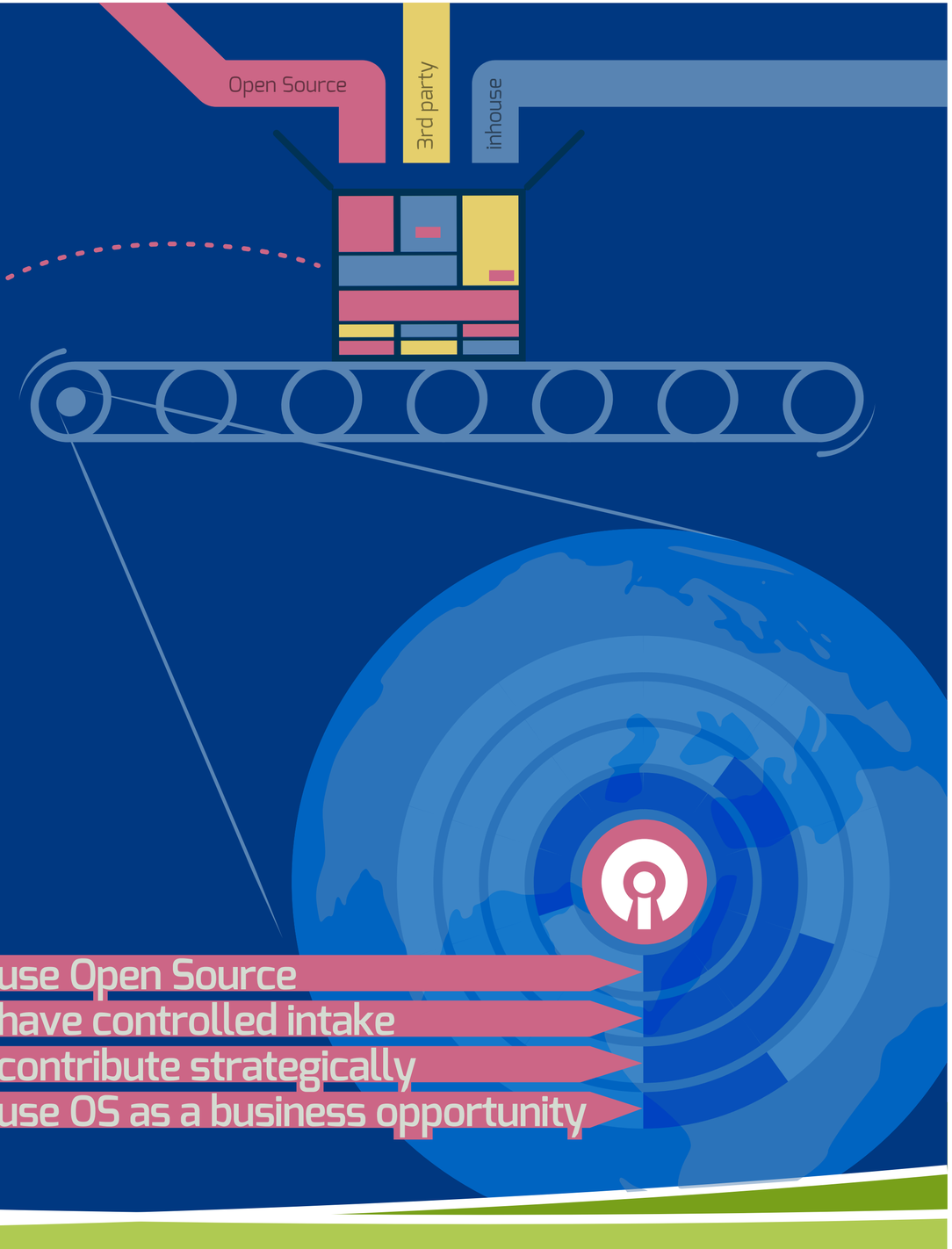
2 Contribution, creating an environment and a strategy for active participation:

- Culture and competence. To impact communities, an organization needs to create an Open Source culture and foster industry experts with leading roles in the communities.
- Product strategies need to include strategies for what you develop yourself (differentiators), what you buy and what you share through Open Source, i.e. Make – Buy - Share.
- Decentralized strategy. A community cannot be controlled from the outside. The developers involved in the community need to actively engage and contribute to drive the direction of the community.

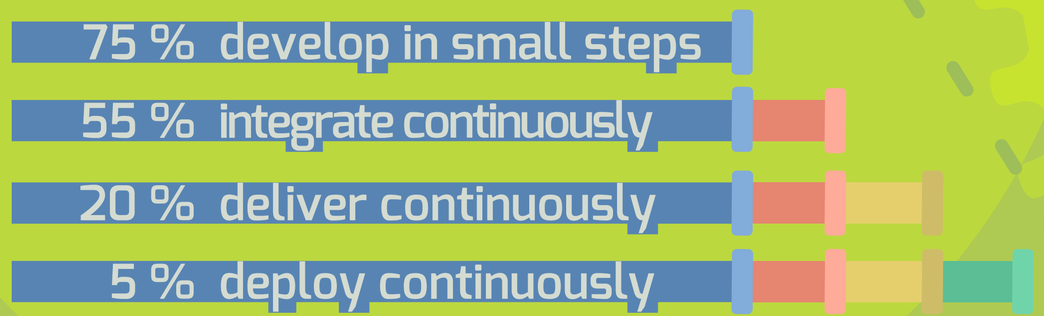
3 Business context, to understand opportunities in the transformation from product value to business value:

- Business involvement. Open Source can be a game changer for the business to enable accelerated growth, disrupt market entry barriers and open up for alternative business opportunities to benefit from new revenue streams.
- Ecosystems. An offering that is part of an ecosystem can become "the business" and not just something that supports the business.
- Servitization. This transition happens when the core product becomes a vital service infrastructure, and thus climbs the value chain.

Status in the interviewed software organization shows that most companies are using Open Source. About a third of them have intake under control (or are on their way), but only a few have reached the second stage where the product offering is made in the context of an Open Source contribution plan. Open source is here to stay, and it is up the companies to decide if they will be a passive bystander or an active driver.



Continuous Delivery



Traditionally new software versions have been released infrequently to customers. The release and deployment process has been painful, risky and time-consuming. Now, more and more companies are turning up the release pace; just consider your smartphone or laptop, not to mention web applications. Working with frequent releases is called Continuous Delivery (CD).

Although CD was initially established to reduce the problems of large and infrequent integrations, companies are now driving the market by deploying new functionality faster. The use of CD has also given other positive side effects, e.g.:

- Increased test automation, leading to better quality
- Drastically reduced cost of each release and deployment
- Faster feedback from the market
- Special patch releases for e.g. security are often not needed
- The whole organization becomes more agile and customer focused

When talking about Continuous Delivery, we are referring to a whole sequence of continuous processes that build on each other and together constitute the complete **delivery pipeline**:

Continuous Development

The code is developed in small chunks that are frequently committed. Committed code has to pass a "definition of done" involving a set of tests, mainly automated. The tests typically comprise unit tests, static tests, implementation tests and code reviews.

Continuous Integration

All code that developers commit is automatically integrated with all the sources in a common configuration branch and then automatically built and tested. Any integration problems are reported back to the developers who committed the code.

Continuous Delivery

The resulting build from the integration is put as a delivery candidate in a staging environment, where it is automatically system tested. If it passes the tests, the release candidate can be deployed into production. The actual deployment is a business decision.

Continuous Deployment

The deployment is automatic, there's no human intervention through the pipeline. Every change that makes it through is put in production automatically. There can be several production deployments a day.

Key properties and enablers:

Short development cycles – By working in small increments,a low-risk delivery flow with a fast feedback loop can be ensured. The goal is that all developers commit at least daily to the mainline, which pushes developers to break down work into manageable pieces.

Build Quality in – By controlling how code is written, reviewed, verified and configuration managed we get control over the flow. Strict definition of done is a key enabler. These practices become the steps in the Delivery Pipeline.

Automation – With automated builds, tests, deployment and documentation, developers get quick feedback on code they have committed. The order of the different automated tests should be organized so that bugs are found as quickly as possible. By automating the delivery pipeline, the flow is enforced - no more shortcuts.

Collaboration – With cross-organizational cooperation and common goals, conflicts about delivery and quality are avoided. Everyone is responsible for the delivery flow and quality. With job rotation and data visibility, collaboration can be strengthened.

CD is in line with the lean software development movement emphasizing optimization of the whole, tying together every part of the delivery process and getting everybody involved in it. Only when you have control over the whole flow can you begin to optimize the quality and speed of software delivery.

Almost all the interviewees consider continuous integration as the most realistic choice for their business. Continuous delivery or deployment would be a realistic option in the case of fully cloud based business lines.

Secure Development

Security has always been important in systems that can be accessed from outside a physically controlled environment. What we now see is that security -- driven by technologies like IoT, Cloud, Open APIs and Big Data -- is moving outside the traditional IT environment into most technical systems, as these become more exposed and integrated. Security is becoming an issue for most development organizations.

- There are at least three essential aspects of security:
- Information Security Management System (ISMS), through e.g. ISO 27001.
- Infrastructure, e.g. firewalls, crypto and APIs.
- Secure Software Development Life Cycles.

Security from an organization (ISMS) and infrastructure perspective has been on the agenda for a long time, although maybe not for technical systems. In this report we have chosen to focus on the third item, Secure Software Development Life Cycles (Secure SDLC in short). Note that there is interaction among the three items, as assumptions about the first two items are used during development.

The advantages of a well-defined Secure SDLC are obvious:

- Security is systematically treated throughout development, leading to better quality and greater security.
- Customers can confirm that security is being taken seriously.
- Cost reduction is achieved through early detection and resolution of issues.
- Business risks are reduced for the organization.

Development of Secure SDLC models is ongoing and there are several good examples: Microsoft's Security Development Lifecycle (MS SDL), NIST 800-64 developed by the National Institute of Standards and Technology, and OWASP CLASP (Comprehensive, Lightweight Application Security Process). OWASP has also developed a Software Assurance Maturity Model, SAMM.

A Secure SDLC adds security-related activities to an existing development phases; some examples are:

Requirements: Threat modeling and security requirements. Security requirements must cover aspects such as how to avoid penetration, detect penetration, isolate and limit impact as well as how to recover from breach.

Design and implementation: Attack surface and vulnerability analysis, security architecture, secure design patterns and coding. A security architecture review using a checklist like OWASP Application Security Architecture Cheat Sheet is recommended.

Verification: Penetration and fuzz testing. Many organizations use external penetration testing services to ensure that they get independent qualified verification.

Release: Secure configuration and environment validation. Ideally the system should automatically validate and monitor the environment to ensure that it provides the required security.

The largest challenge in establishing a successful Security SDLC is integrating security activities in the existing software development process and organization. The complexity can appear overwhelming and give the impression that productivity and flexibility will be reduced. Yet, done correctly, they can improve the system through better general quality at a moderate cost.

Some key success factors to work efficiently with security during development:

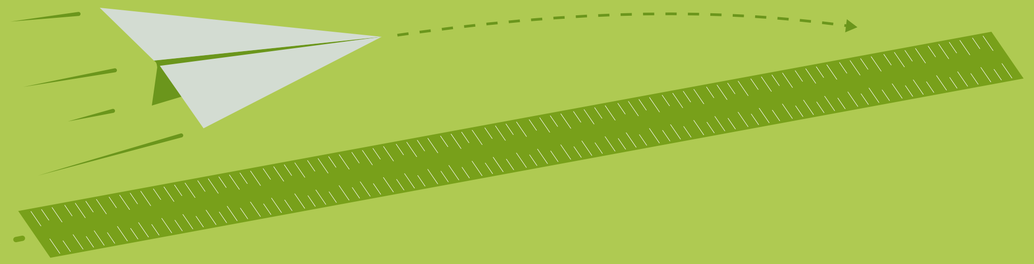
- Use the public Secure SDLC's as inspiration. Integrate security activities in the organization's existing development process, as the existing process suits the needs but is weak in security.
- Let development teams and not security experts manage all security related activities. There is still a need for security experts but they should provide support instead of driving their own activities. OWSP provides practical checklists.
- Perform security activities at the right time. This is increasingly important and challenging as we move from traditional waterfall development to agile development. The Microsoft guidelines provide good examples of what activities to do and when.
- Security has to be part of development of all features, not only specific security features. However, since all bugs potentially have security implications, the positive outcome is overall increased code quality.

The interviews indicate that the security maturity in the manufacturing industry was low, but is rapidly increasing. They have been accustomed that their systems are in a secure and controlled environment, but this is not the case anymore. The largest problems are lack of security competence, security activities and co-ordination across the lifecycle.

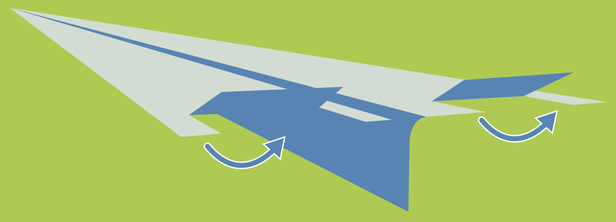
70% lack security competence
40% lack awareness of security risks

70% lack security activities in development
75% lack coordination of security activities

Data Driven Development



$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



45 % validate and analyze data
20 % deliver continuously
10 % experiment actively

In today's world, companies of all sizes and in all industry segments are collecting an ever-increasing amount of data. In the last couple of years there has been a growing interest in how data can help redefine product development processes to the level of Data Driven Development (DDD). In short this means that instead of basing product development on expert opinions, development is directed by a feedback loop with the customer.

Thus you need to ensure that data collection and analysis are part of the process, so that system and user responses can be measured. The drivers for you are higher customer satisfaction and better understanding of the market and the merits of the product.

The main enablers for doing DDD are the following:

- **Iterative development,** to develop and deploy frequently, so techniques like continuous development will increase your ability to experiment.
- **Data collection.** The ability to collect relevant data in many different shapes and fashions:
  - **Qualitative data** can be retrieved from users expressing their views about a product through e.g. user surveys, ratings or social media – or even video recordings of consumers using the product. Understandably, this type of data requires careful consideration in the analysis step, but is often vital.
  - **Quantitative data** are often collected through automated systems, like code probes that track user actions, system response and diagnostics or third-party market data.
- **Data storage and validation.** You need somewhere to put your data, and you want to secure the quality and validity of them, including proper handling of privacy and security.
- **Data analysis.** Most crucially you want to derive insights and knowledge from your data. This is not only about statistical methods, but can also involve understanding user feedback and psychological and behavioral factors. Ideally you would like your development team to be able to do its own analysis, but expert support may be needed to get started.

With this in place, an organization has the tools to seek answers to more or less any question or validate any assumption it might have – and to take action accordingly. DDD can be taken one step further by reshaping the creative process into an experimental approach. This requires an ability to break down product ideas into iterations and experiments and to ask the right questions.

Your experimentation could be about: – New product ideas and how to hone these into something that fulfills real needs on the market. – Evolving existing products with new features by ensuring that each new thing you add makes sense to your customers and creates real end-user value. – Pruning (or removing) products by examining the actual use of the current feature set.

It's not a coincidence that many of the big players in the tech industry are using DDD extensively. They have found that it is a key to ensure user acquisition, drive increased customer satisfaction, and establish a market and customer driven innovation system.

DDD will be one of the major changes in the years to come As with e.g. Agile and Continuous Delivery, it also requires a transformation of culture. All parts of your organization need to agree to the methodology and work much closer together than they are (probably) used to, e.g. the product planners with the engineer, all having customer data and experimentation as a main vehicle for innovation.

The status in today's software organizations is this:

- They collect (huge amounts of) data – but do only limited validation and analysis
- They make quick updates when needed – but do not build a continuous, cyclic (innovation) flow (including Continuous Delivery)
- They ask what the consumers think – but don't experiment actively

Servitization

In essence Servitization is a transformation journey – it involves companies developing the capabilities to provide services and solutions that either supplement or replace their traditional product offerings. Recent technological advances such as cloud computing, big data, mobility and social media have enabled this. Strictly speaking, Servitization is not a software methodology, but it is fundamental in the current transformation of the industry, and heavily dependent on software.

Business drivers for services:

- Customer expectations. Customers become increasingly demanding and organizations are pressed to adjust to those high standards, and thus implement customer centricity.
- Financial incentives. Shrinking product-based profit margins are spurring the need for service-based revenue growth. Revenues from services are greater than new product sales, especially in times of economic crisis.
- Gaining competitive advantage. Customer service has become a competitive trump card, making services are difficult to imitate and locking out competitors.
- Marketing opportunities. Use services for selling more products. By offering services, companies gain insight into their customer's needs. This insight can be used in data driven development but is not used extensively except in a few internet companies.

A key to successful servitization is to create a "win win win" situation for producers, suppliers and customers. The customer can see advantages in a better cost structure, reduced financial risk and operative cost, and ideally can see a common goal with the service supplier.

There are several levels of servitization:

Base Services (products and spare parts)

- The customer owns the tangible product.
- Examples of services offered: product and equipment provision, spare parts provision and warranty.

Intermediate Services (maintenance, monitoring, field support)

- The customer owns the tangible product.
- Examples of services offered: scheduled maintenance, technical helpdesk, repair, overhaul, installation, operator training and certification, condition monitoring and in-field service.

Advanced Services (pay per use)

- The tangible product is owned by the service provider or financial partner.
- Examples of services offered: customer support and rental agreements, risk and reward sharing contract, revenue-through-use contract.

Fully servitized (independent of product supplier)

- The service is the main offering, and can be connected to products from any manufacturer, owner or operator.
- Examples of services offered: Typical examples of fully servitized companies are Airbnb and Uber. Even streaming services like Spotify, Netflix and YouTube can be considered part of this category. We now see Netflix is starting to produce its own content and Uber is developing self-driving cars: as the pure service providers become more powerful they start tapping into the other parts of the ecosystem.

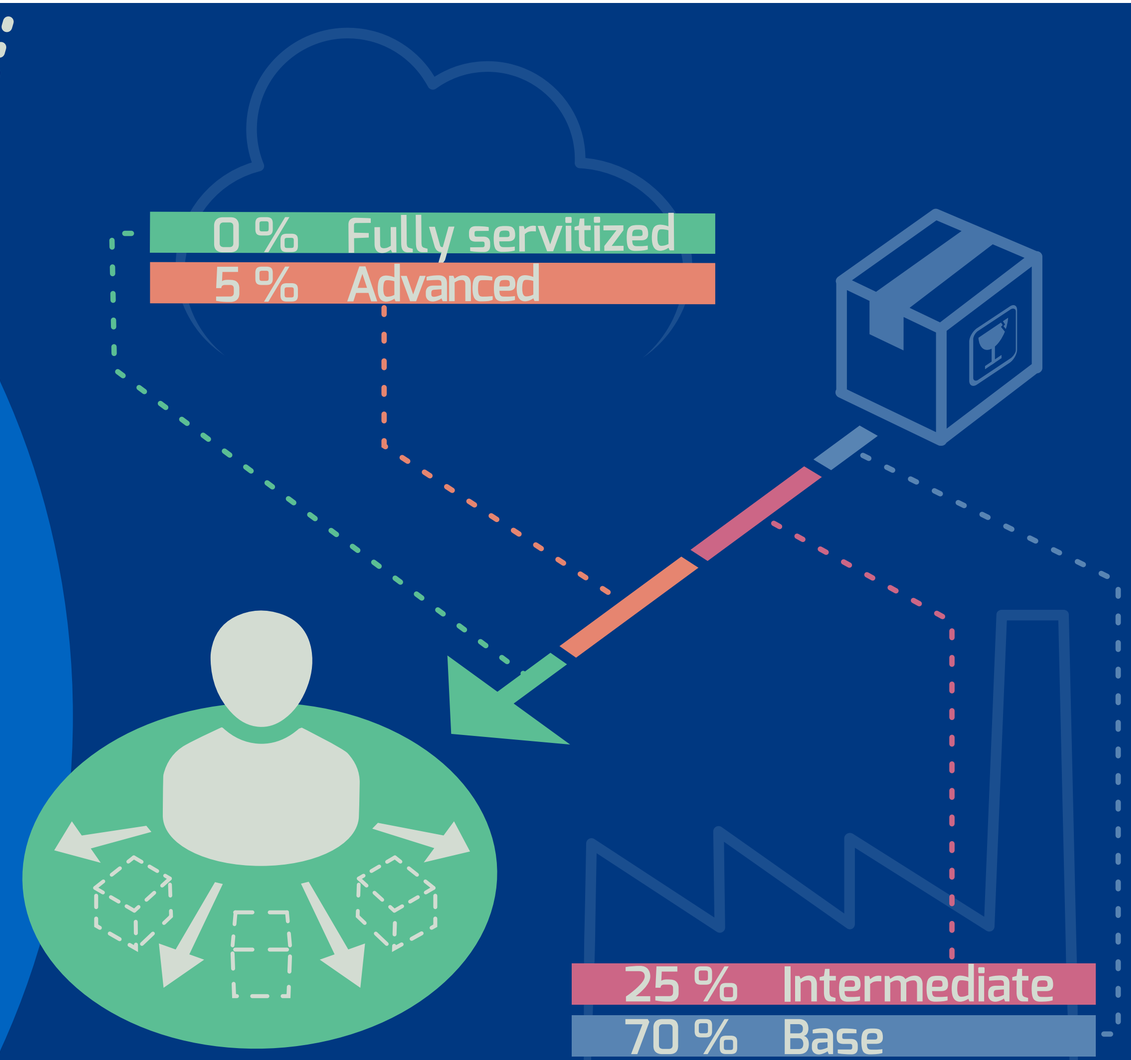
There are many challenges to servitization for a product company, both internal and external:

- The organization was set up to produce physical products and does not have much competence in providing services. Most managers have hardware and production focus.
- Servitization requires new competences, especially software resources, which may be scarce.
- Moving to servitization requires a holistic view on the entire value chain and has to consider organization, culture, budget and KPI's.
- The existing ecosystem of partners is threatened by the product company taking over some part of their job.
- Customers are reluctant to be tied too closely to one supplier.
- Technical challenges related to servitization, such as privacy, security, usability, big data, availability and performance.

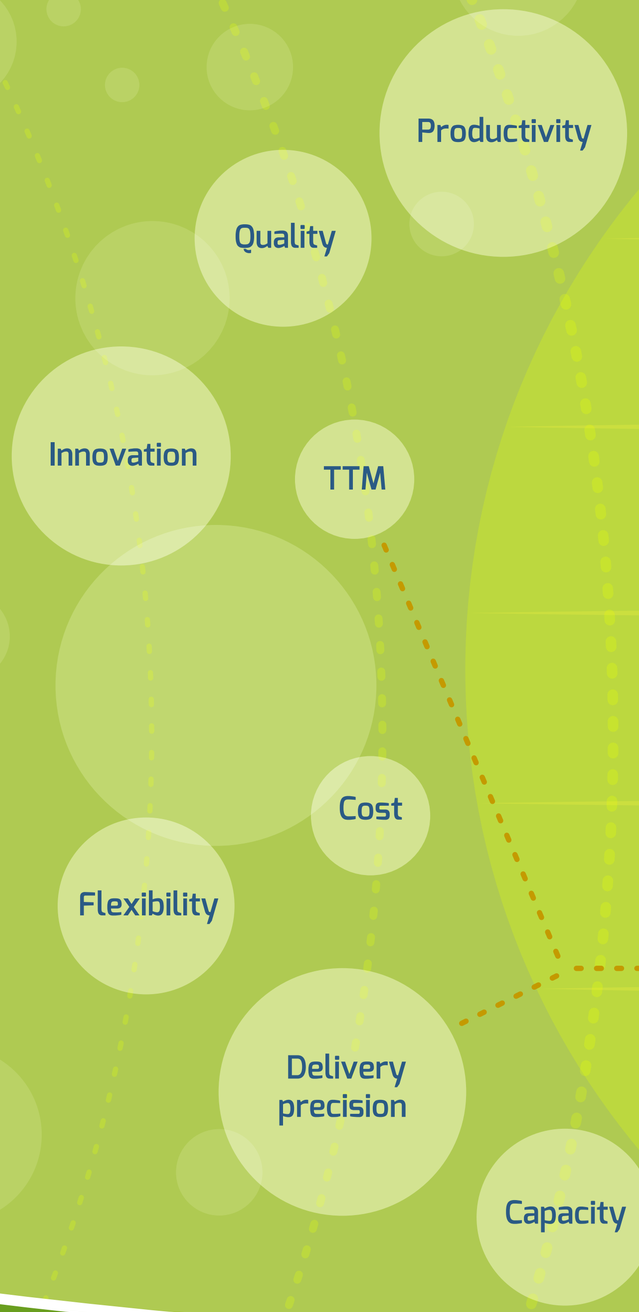
All of the other software methodologies presented in this report are useful or required to support servitization:

- Secure development will ensure that the services are not hacked and valuable customer data is not stolen.
- Data driven development reveals how services and products are used and facilitate meeting market needs faster and better.
- Open source frameworks are dominant in the web domain.
- Adapted agile and continuous delivery are almost a must when moving into services.

From the interviews, we can see that servitization is coming, the question is just who will get there first.



Conclusion



Even though the companies interviewed vary in size and operate in different domains, there are some very interesting trends:

- Increased connectivity and IoT are making domains more similar, i.e. in theory, all companies can now be in direct contact with their products, collect data and deploy changes continuously. Previously this was mainly restricted to pure Internet companies.
- Large-scale development is spreading. It started over forty years ago in domains such as defense and telecom. Now more and more areas are facing these challenges; for example, the automotive industry has faced exponential growth in areas like infotainment and active safety and now in autonomous driving.
- Many of the established, large development teams have in turn started to learn from the fast-moving development teams in Internet and start-up companies.
- Mandatory process standards, e.g. for safety, security and maturity, are emerging in more and more domains.

The trending solutions are supporting these changes, and we see streams of knowledge in transfer between domains.

Several of the trending solutions are also tightly connected. This concerns in particular Adapted Agile, Continuous Delivery, Data Driven Development and Servitization. Solutions can be selected based on the business drivers and implementation adapted to the needs of the company.

So how are our companies doing then?

Judging from the responses, there is room for improvement.

Challenges experienced included:

- Chosen solutions are often not driven by business goals.
- Existing good practices are sometimes missed, as software process improvements often are led by a process group separate from the projects.
- Taking on too many improvements at once; better to focus on a few.
- Many organizations aim high, but still struggle with basic software engineering practices like requirements and planning – you must walk before you can run.
- The human side of change is neglected. Change starts with people and then continues in organizations.

These challenges often lead to the fact that many solutions only get partly implemented, if at all.

Inevitably we have not seen the last changes in software development, far from it. Organizations must get used to the continuous change and learn to adapt faster and faster.

In other words

As market expectations continue to grow, organizations must:

Do the right thing – and it's not just about being effective. They must be aware of coming development trends and anticipate them in time. How, for instance, will AI and machine learning affect software development and verification? A game changer, most certainly.

Do it right – even when doing the right thing, being utterly efficient has never been wrong. This report asks what's cooking in the industry. As in cooking – apart from having the right ingredients and an aim to make a good dish – you also need the right tools, processes and competence to succeed.

To wrap it up

We asked: what are your business drivers?

There are so many solutions that it's easy to get confused. This is in fact the key lesson: to stay aware of your business drivers, determine what is needed to succeed and consistently implement your strategy.

Again, as in cooking – too many cooks who don't work together will end up making a dog's breakfast.

Method

Results of approximately 20 interviews carried out by Addalot during 2017.

Interviewees included software development managers and professionals in IT organizations at software product development companies in the automation, automotive, IT, med tech and telecom industries.

The interviews combined a quantitative questionnaire with qualitative open discussion.

Addalot transforms companies where system, software and IT are business critical in order to meet their objectives.

Our services are based on 25 years of experience and include assessments, transformations, interim management, specialists and training.

Learn more about us at:

www.addalot.se

addalot