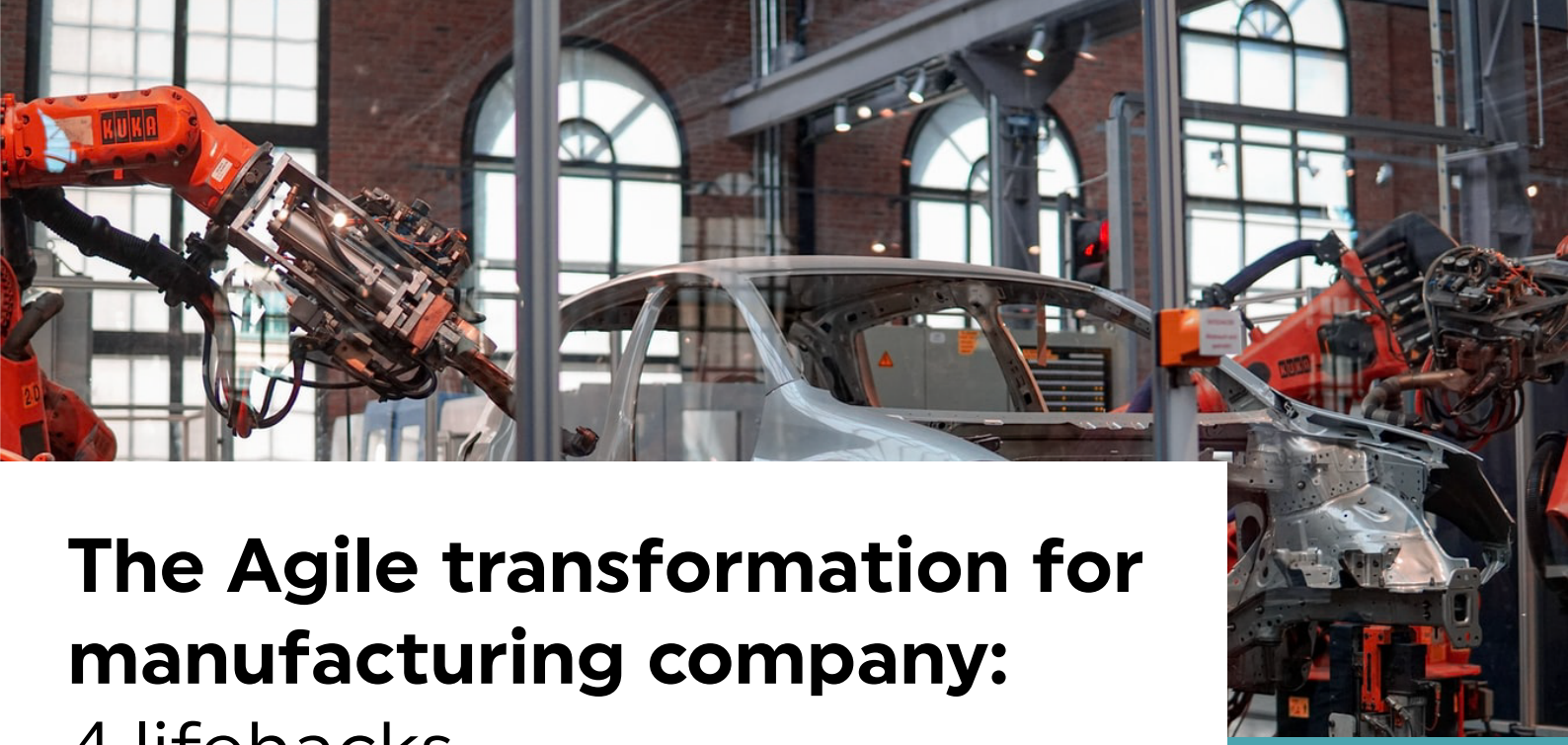
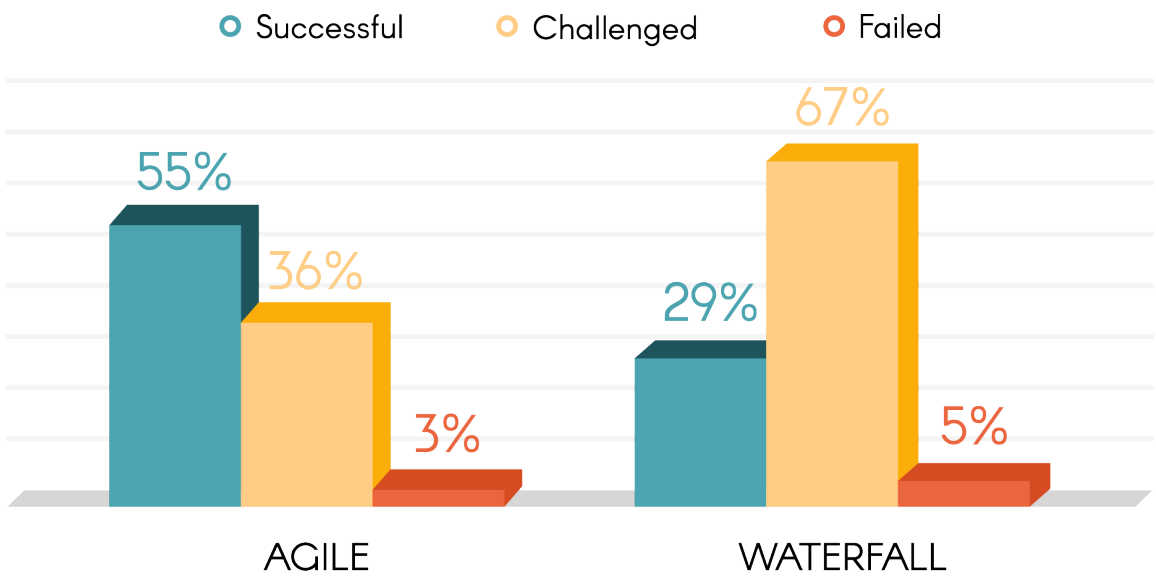


The Agile transformation for manufacturing company: 4 lifehacks

devvela

The last two decades have offered plenty of proof that applying the Agile approach to developing applications leads to dramatically more successful results than the Waterfall approach. The Agile methodology boosts every important indicator of programming output. Additionally, the quality of the product improves. Our experience shows a two-fold drop in costs and four-fold time-to-market savings for companies when using this flexible approach and set of tools.

Why then do some manufacturing companies choose not to pivot away from the ineffective Waterfall approach?



This has to do with the predictability historically rooted in manufacturing companies' DNA. All stages of creating their product are always based on an unambiguously interpreted model and each step must be performed in a clear and predictable way. You have to be able to predict per-unit costs, expenses for components, marketing, sales. But developing custom software is fundamentally different from producing industrial goods, and the Waterfall approach (fixed price, fixed volume, fixed schedule) simply does not work here.

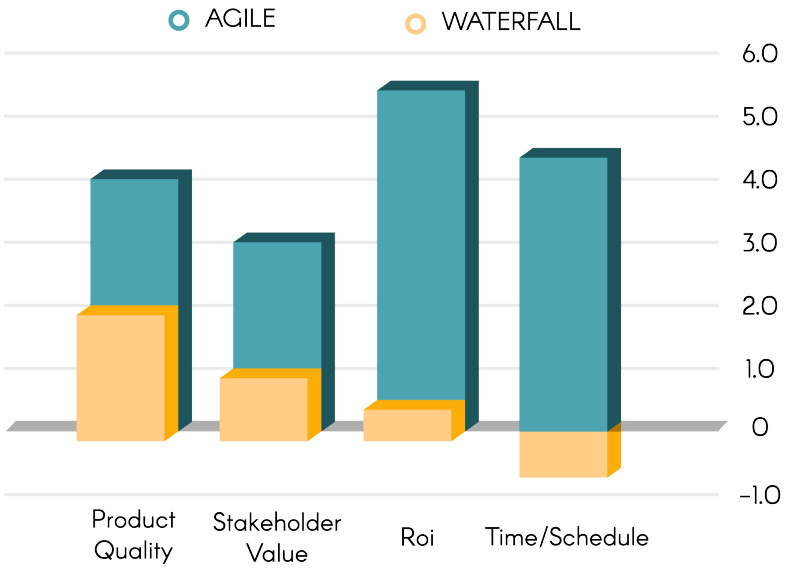
Drawn-out creation of project documentation, long periods of time for approval, developing software without any contact with real-world users — all this means that for a significant time while a business is busy bringing the project to life, no actual value is created for the business, because the company is not producing functional software. When the programming is done, it turns out that what was developed is not exactly what the business really needs. Moreover, the market is never standing still. Requirements can change during the development process, yet the software does not take these changes into account.

The Agile approach, on the other hand, allows you to deliver a product quickly, at high quality and meet the demands of the business and the market.

Bespoke software is something that a manufacturing company might need in the following situations:

- Developing a new product to find gaps and opportunities in their customer's experience.
- Bringing outdated software up to date
- Process automation

When it comes to any of these situations, a quick solution to your problem and immediate business results is only possible with the Agile approach.



In this white paper we will talk about 4 practical applications (or lifehacks) that are essential when beginning to use Agile to transform your business:

- The baby-steps approach. Developing a product in small pieces.
- Gathering flexible requirements for rapid development.
- Lowering your expenses for quality control and support by using continuous integration.
- The Build – Measure – Learn cycle. Collecting live user data.

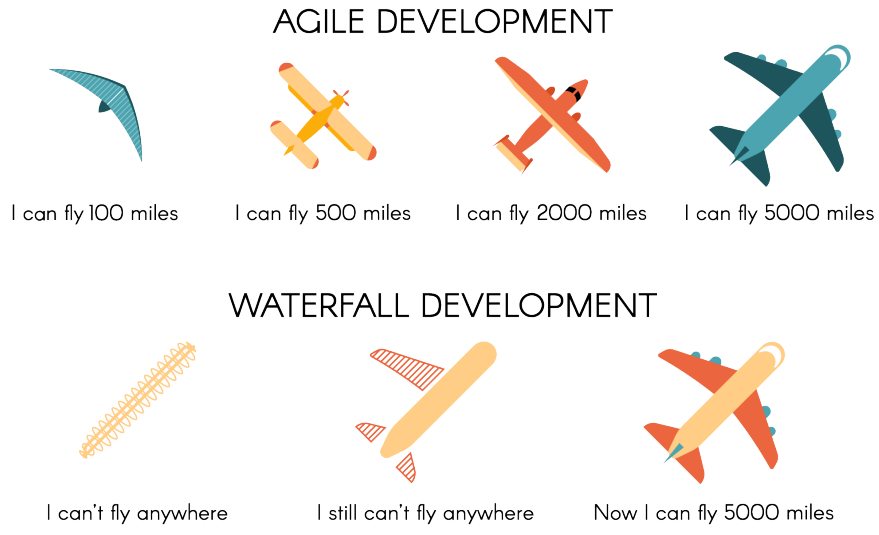
PRACTICAL APPLICATIONS

The baby-steps approach | Flexible requirements | Continuous integration | Build – Measure – Learn

1. The baby-steps approach. Developing a product in small pieces.

Great success begins with small victories.
Immediately applying Agile methodology (an approach new to your organization) to a large and complex project is not the best idea.

As you implement Agile in your enterprise, the principle of "divide and conquer" will prove more effective. You must choose some task from your product-development plan and break it down into the smallest possible pieces



Let's discuss what size project is optimal for beginning to work with iterative delivery. Let's start from the size of the team involved. In short, less than 850 man-hours (per month) is probably not enough. On the other hand, anything more than 3000 man-hours (per month) would be too risky.

Now let's consider these figures in detail.
For that, let's take a typical (small) team consisting of:

- Product manager (PM)

The product manager links the business requirements for the product with the Minimal Viable Products backlog starting from the first iteration.
The PM's role is very important but, unlike the developers, he or she is not 100% involved, in terms of working time, in the actual day-by-day delivery of any one product. PMs generally manage delivery of 2–4 products (depending on their size and complexity). So, for our appraisal here, we will assume that the PM is spending 25% of his or her total working hours per week on the product, i.e. approximately 9 hours.

- Developer 1 (100% involved in delivering; 35 hours per week)
- Developer 2 (100% involved in delivering; 35 hours per week)
- Designer (40% involved in delivering; 14 hours per week)
- QA engineer (40% involved in delivering; 14 hours per week)

In total, this team of 5 people (a portion of whom are only part-time involved) spends 107 hours per week on the delivery (i.e. ~3 FTE). Development is carried out in two-week sprints, and the team spends 214 hours per sprint.

The effectiveness of Agile can be seen as the team iterates over the product, creating small functional parts thereof and then testing them with customers and tweaking them (and thus the product as a whole) as development progresses.

In our experience, to get a functional end-product, a team must iterate (i.e. operate in Build–Measure–Learn cycle mode: generate ideas, swiftly implement them, test them and tweak the product based on the data and any new ideas that then make their way into the next iteration of the cycle) for at least 4 sprints of 214 hours each. Thus one can say that delivering a product to market within the Agile framework takes at least 856 hours.

You will encounter many skeptics in your industry who seriously claim that corporate and industrial projects are difficult to break down into small components. That is just not factual.

Take a careful look again at your projects and you will clearly see that each product, and each technical implementation, can be broken down into subcomponents ideally suited for iterative delivery.

2. Gathering flexible requirements for rapid development

Waterfall, business analytics, documentation running into the hundreds of pages, Gantt charts.
All of that is so very copious yet... useless. Sounds familiar?

A minimum of six months are spent on collecting, analyzing, and formalizing data for a future product. The result is a thick stack of pages rigorously setting out the structure of the product, the draft documentation for web services and mobile applications, and the requirements for the design of the service and the user interfaces. The stages and deadlines for manufacturing are effectively laid out in aesthetically pleasing Gantt diagrams: each stage has clear dates for starting and ending work on it, each stage gives way to the next, and the finished product is ready for delivery just in time.

Everything seems to be in order. What could be wrong here?

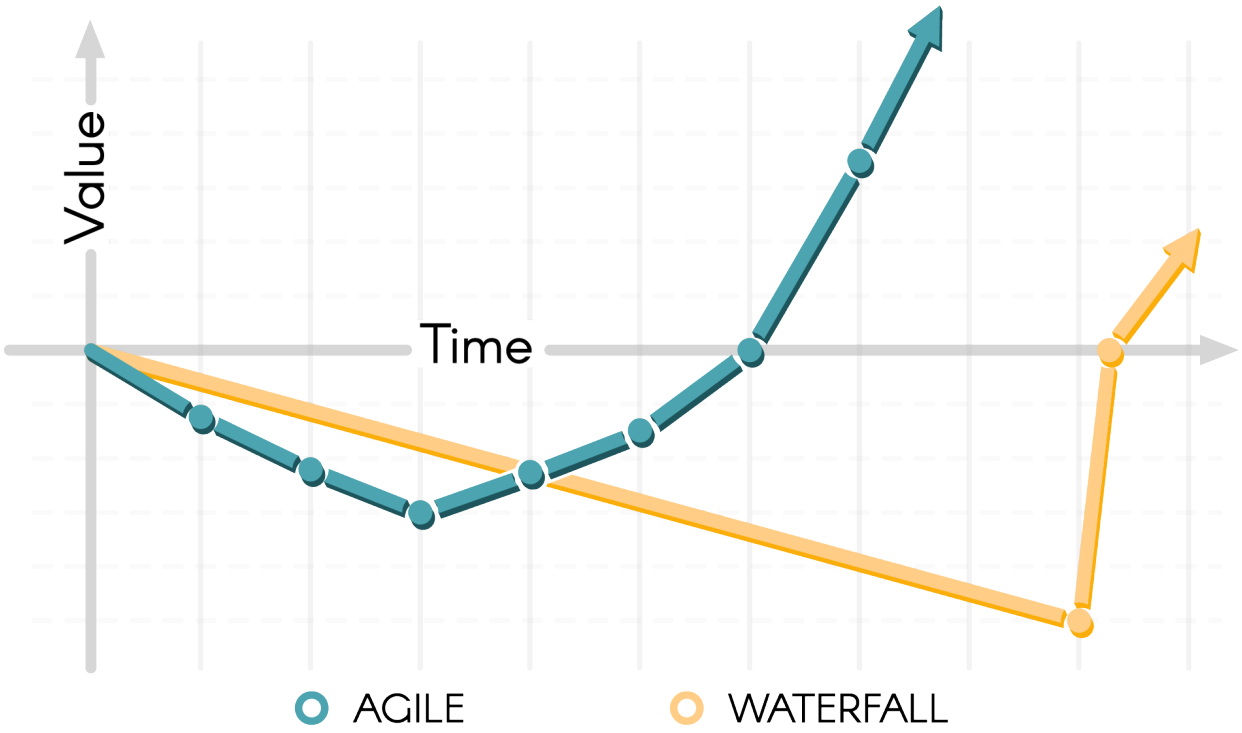
There are two problems.

Problem number one. Six months have gone by, but in spite of the mighty tomes of documentation and intricate charts, no actual value has been created for the business. The company has come no closer to a functional software or app.

Problem number two. All these charts are just not going to work. And no one is going to read all those pages.
The real-world requirements dictated by the market will have changed even before you finish drawing up the documentation. That perfect picture with stages beginning and ending on a strict schedule, will break down when it meets reality and the first change has to be made to the stages. In all honesty, deep down inside you are already aware of all this.

How can you avoid this six months' standstill when no value is delivered and your software starts to lose its edge, before development has even started?

Answer: you need to try a flexible set of requirements, the goal of which is to maximize your product's consumer value while minimizing your losses.



A crucial thing here is avoiding a situation where key technical and user-interface design decisions are drawn up by a single person, who works without the involvement of a cross-functional team, and t without the knowledge that the team members have.

Flexible gathering of requirements is something that only a group of people can carry out.
The goal is to reach a common understanding among the entire team of what the end results should be, including the needs of stakeholders and customers. The group needs real working meetings (COVID has lent legitimacy to online meetings). The duration of these meetings ranges from 4 hours to 2 days. During the meeting, the participants define the personas and the user stories characteristics for those personas (grouping what functional blocks can be obtained for the product), and they create an outcome-based product roadmap.

Personas capture the characteristics of users that have some goal, that is, a problem that must be solved or a benefit that your software should provide. Any new functionality added to the product must now be defined in relation to how a particular persona acts.

User stories are brief statements of intent describing what the product should do for a particular persona. They represent small fragments of valuable functionality that can be implemented within a single sprint.

User stories are brief, easily readable statements that the team, stakeholders, and users can understand. User stories are easy to define, and it is also easy to judge the effort required to implement them. To write user stories you do not need documentation running into the many pages. Instead, they are organized as lists (backlogs), which are easy to order and reorder as new information comes in.

The team further groups user stories into a backlog/release (sprint) plan for delivering the product. The release plan should be based on outcomes, not on outputs. That is, the result of each sprint should be a functional part of the product, and not a mere list of completed tasks that are not actually usable in practice.
Creating a release plan happens in three stages:

- prioritizing user stories;
- choosing a limited set of user stories for each sprint (higher-priority user stories go into an earlier sprint);
- defining the target outcome for each sprint.

The flexible requirements represented as user stories and grouped into a release plan, make for a shared understanding among the members of a cross-functional team. Each team member feels like a coauthor of the product, since they each contribute their own ideas and are involved in really delivering the product.

Thus, having a meeting to define flexible requirements that precedes the start of development, is quite effective: you will spend 1–2 days, but the team will have enough information to begin working and to supply functional software.

Before the start of each sprint, the array of user stories to be implemented within it is reviewed by the team (together with stakeholders) depending on the results of previous sprints, any new understandings of the product, and shifts in priorities in the backlog. A change in requirements, like any new input, can easily be updated in a central live repository that every member of the team has access to.

In summary, you can say that a flexible set of requirements helps bring forth working software instead of mere comprehensive documentation, while also responding to real-world changes instead of following a strict plan that may no longer be relevant. These are two of the four basic cornerstones of the Agile manifesto.

3. Lowering your expenses on quality control and support by using continuous integration

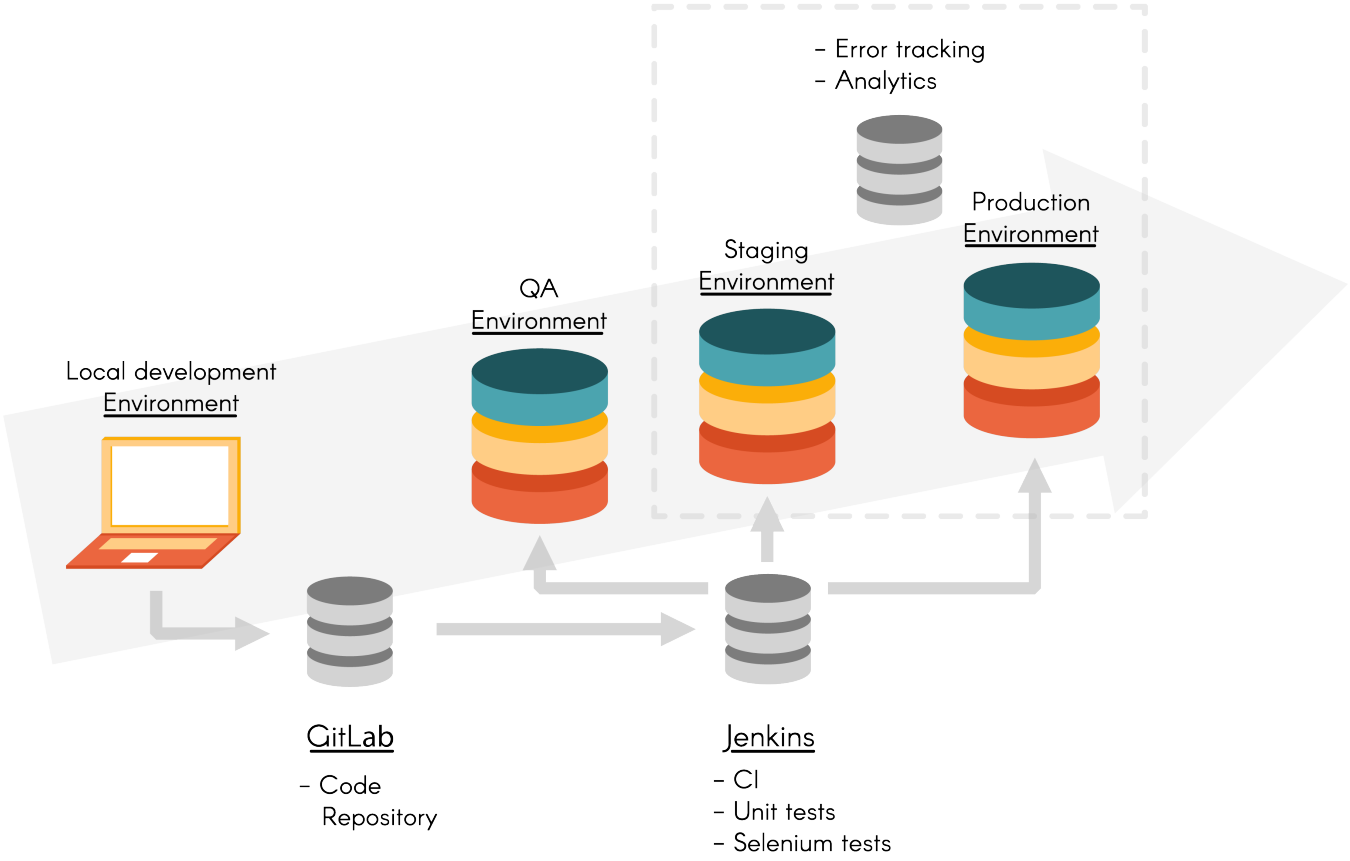
The usual scenario for product development might look approximately like this: engineers work on local computers and check the code into a source repository (Git or similar). Each developer creates their own part of the product locally and tests their own code out.
The serious problems come when you start the code-merging and testing phase. A heap of conflicts arises, integrations don't work, the bugs or defects found by your testers are set down in opaque Excel documents. Consequently, the deadline is shifted forward for this product that everyone worked on individually, and the launching of actual functionality is postponed to the last phase of development.
This is where a major hitch arises: no explanations or forecasts for the cost and time needed to fix defects, are going to pass muster with the project's sponsor. And all because these problems/defects/clashes in integration were detected after development was completed and the budget had been depleted, while no actual functionality has been supplied. It is not clear what can be done now.

To not end up in such a situation, one must use a development methodology of continuous integration (CI) and a set of tools that allow engineers to integrate their code daily into a common repository. The CI platform runs automated tests and builds the product, allowing members of the team to identify any defects and inconsistencies immediately after the source code is checked in.

The products you are making should have separate environments for development, code merging, testing, demonstrations for stakeholders, and production.
These are essential, and if you do not have the proper conditions for all this yet, then you need to create them: invest in infrastructure in order to create an environment for gathering code and implementing continuous integration. The use of cloud solutions will allow you to avoid large equipment costs for your infrastructure, assuming of course that your business's security policy allows this.

In any event, a CI setup imposes certain costs towards specialists with special skills, equipment, and software licenses. However, these costs will be much lower than what you will earn in terms of reduced testing costs, a better-quality product, and faster delivery.

The entire environment can be configured, e.g. like this:



Once more on the advantages of CI.

1. *The code merging is conflict-free:* the team uses a single repository to which a team member commits his or her code at least once daily.

2. *At day's end the CI platform prepares a nightly build of the product in the code-merging environment and performs all automated tests.* This allows you to be sure that when product builds are prepared in the testing, demonstration, and production environments, there will be no problems and everything will go through automatically.

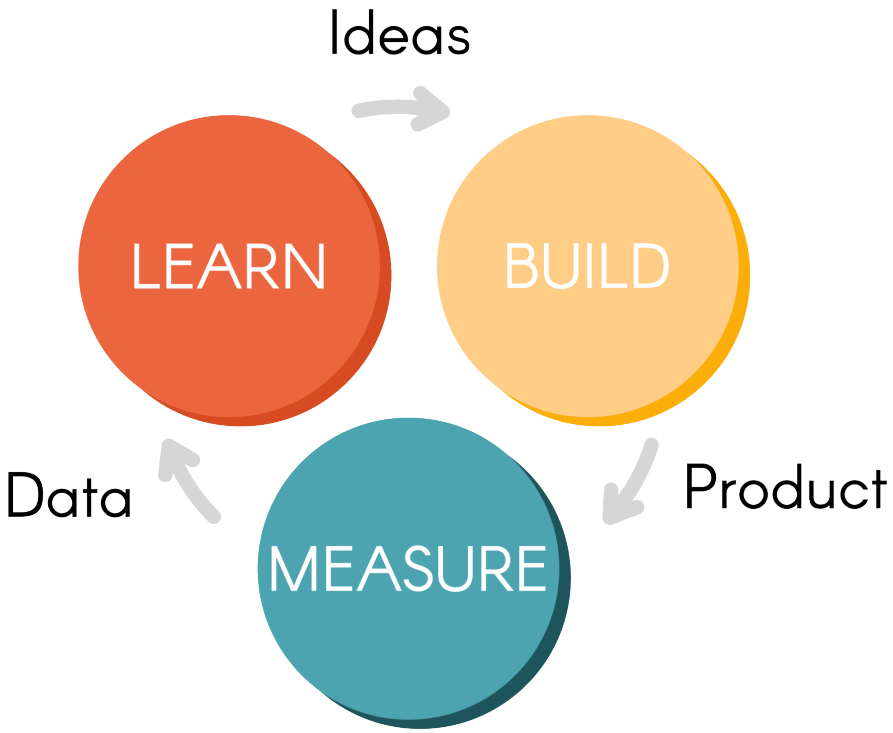
3. *By using continuous-integration methods, you lower your expenses on quality control and support,* because the CI platform automatically conducts a maximum of tests without team members having to intervene, and it immediately issues recommendations on fixes to be made.

4. The Build – Measure – Learn cycle. Collecting live user data.

Very often companies spend months, sometimes years developing and perfecting a new product. Yet all that time they do not show the product to a potential buyer, even in just some basic form. They do everything quietly without attracting attention from consumers. So, after a year of painstaking work, such companies bring out the final version of their product, only to be met with disappointment, as it is practically impossible to get wide takeup from customers, who had never been shown the product before. Even if customers begin to use the product, it turns out that they do not like it much, the product does not meet their needs, etc. As a result, the product fails in the marketplace.

Avoiding such disappointing outcomes is something that the Build–Measure–Learn feedback cycle can help with as you develop your product.

The Build–Measure–Learn cycle includes the following stages of work on the product: creating small functional parts out of the overall product; testing these small parts out with potential customers and judging their reaction; and learning lessons from the results of testing and making changes to the product, with the aim of better meeting customers' expectations (see the image below). The aim of Build–Measure–Learn consists of continually improving your product during the development process, so that you bring to market precisely that product which your customers demand.



In practice, this means that the Build–Measure–Learn cycle becomes an important part of your two-week iterations, or sprints. In essence, during each sprint the team creates, tests, and demonstrates specific functionality of the product. Through constant evaluation learning from it, the team can adapt more productively, as well as gain clearer business requirements for achieving the set goals. Thus, all interested parties can test out and improve an actually working product on the fly.

As a result, the business gets a product that customers need, exactly what the business was counting on. In addition, during this process, the team generates creative and technical ideas and tests their viability, thereby improving the product without impacting the overall budget.

Launching a new product is always fraught with a great deal of uncertainty, and the Build–Measure–Learn approach is optimal for testing out a hypothesis within such an environment. A company that rolls out the Build–Measure–Learn cycle in the making of its products, learns faster than its competitors and creates products with real value and measurable business results in the competitive marketplace.

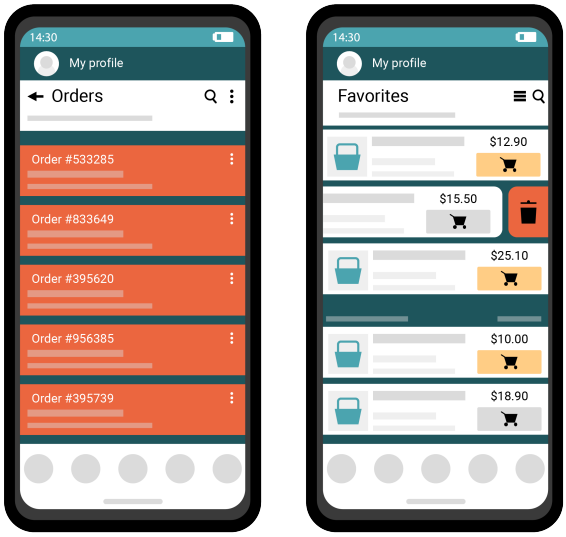
Begin using Agile effectively: start with your customers

Naturally, introducing Agile into practice is a vaster subject than we have touched on in this white paper. Certain topics have not been dealt with here, whether general ones like how methodology changes might be linked to changes in the company's structure and culture, or utilitarian matters like the types of meetings arranged by the Agile team and other stakeholders when preparing for sprints and carrying them out, approaches to project budgeting, and developing metrics for delivery.

This is not about an exhaustive description of the Agile philosophy and techniques.

Something should serve as the starting point for changes within your company. The four principles outlined here are essentially changes in processes, and to start using them does not impact the architecture of the company's main platforms. It is something that is safe, the first baby step towards transforming your business through Agile. However, this step alone can already have an impressive effect in terms of reducing costs and speeding up delivery.

The key thing is to choose initiatives from among your company's product-development plan where you can start to use the Agile approach. Again, do not delve into the depths of your manufacturing process. Choose initiatives that are focused on your clients, for example optimizing your electronic front office. Especially suitable are projects for developing online self-service or personalized customer service, as well as mobile and web applications that simplify the ordering process for your products or allow customers to receive needed information in a form convenient for them.



Mobile app that optimizes basic ordering process

An app that helps business clients to reduce the number of iterations furing the everyday ordering process
+10% to average order amount

About Devvela

We are a custom software development firm, delivering data-driven products that provide better engagement and interaction for our client's stakeholders and customers. We plan, design, engineer, transform, optimize and automate to solve real business objectives for manufacturing companies focusing on product development, process automation and legacy software transformation.

Devvela Inc

159 N Sangamon St, 2nd floor
Chicago, IL 60607

https://devvela.com

devvela