Topic: DevOps

Topic: DevOps

Lesson 1: Basic Explanation of DevOps

Lesson 1: Basic Explanation of DevOps

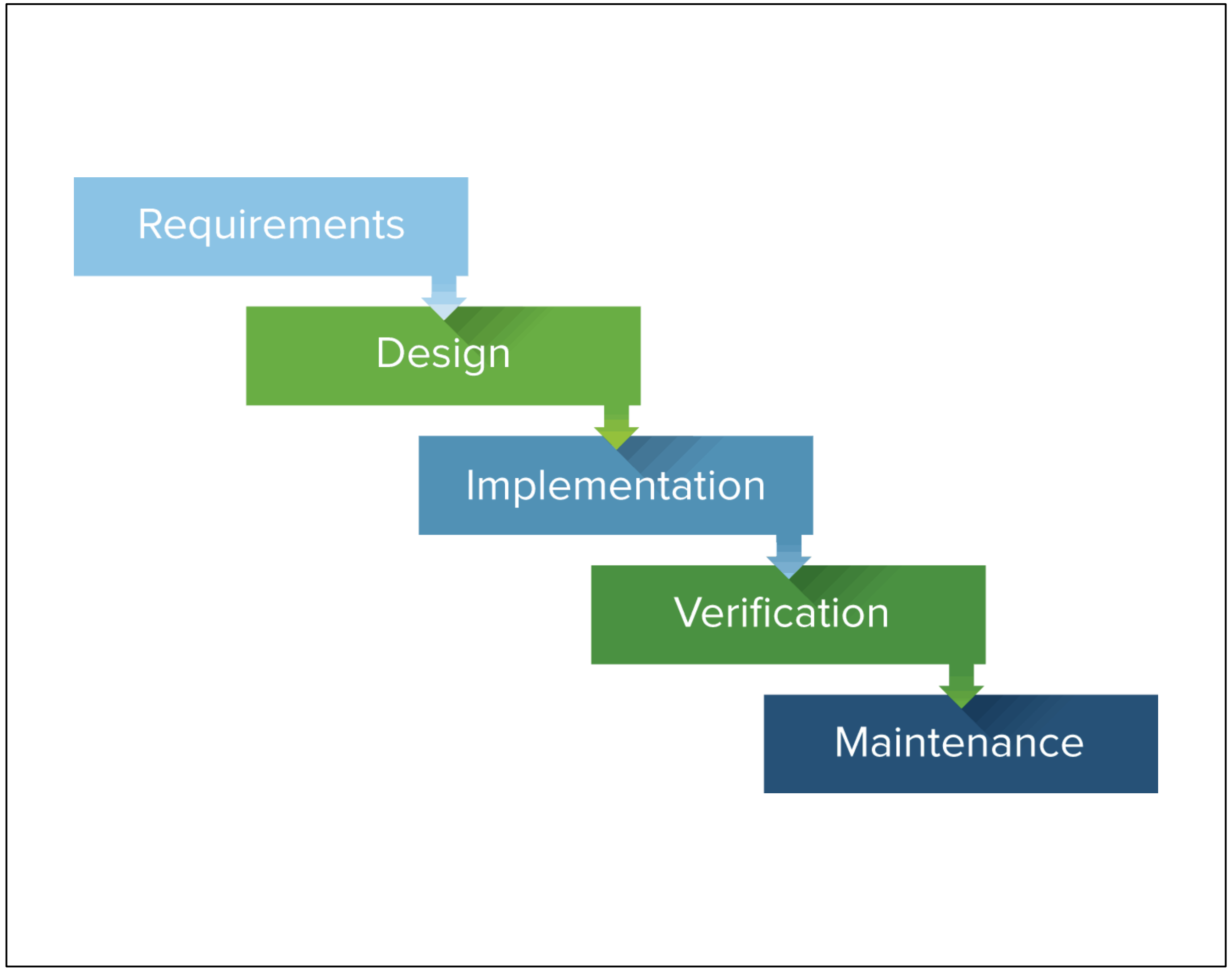
So, you might be here because you are wondering what is meant by this buzzword *DevOps*. To give you a definition, DevOps is *"… a culture shift designed to improve quality of solutions that are business-oriented and rapidly evolving and can be easily molded to today's needs"* [1].

Okay. So we are talking about a culture shift, the improvement of quality, business-oriented solutions, and addressing rapidly evolving needs. Sound good. But what does it really mean? And how do we bring this into practice?

Lesson 1: Basic Explanation of DevOps

Let’s first take a look back into history. Back in the days, before DevOps, software development at AFKL and many other companies happened according to the waterfall methodology. The waterfall methodology consists of the different software development phases, which happen in linear sequence. This means that the next phase only starts once the phase before it has been finished.

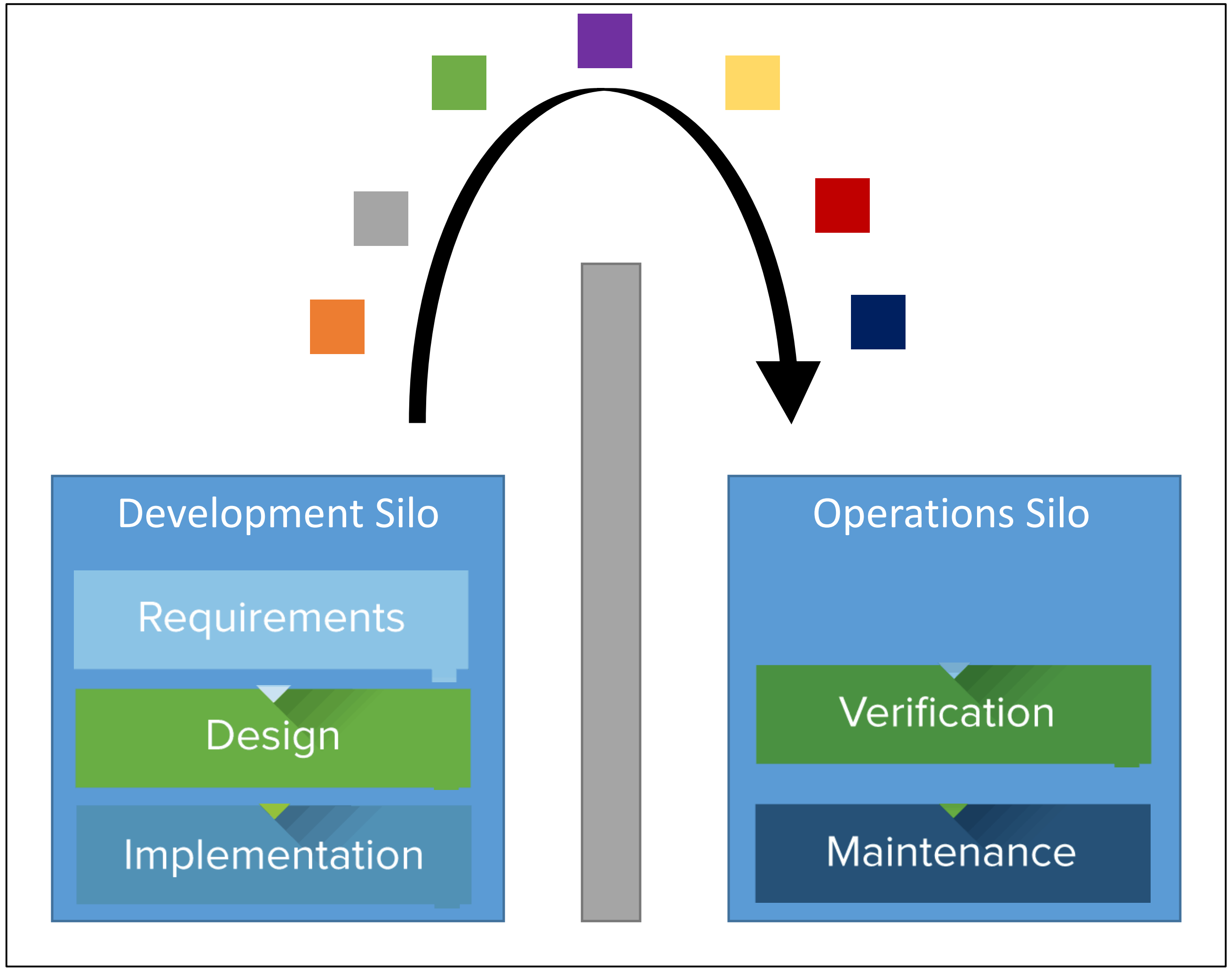
The different phases of the waterfall methodology are the requirements phase, design phase, implementation phase, verification phase, and the maintenance phase. Go to *Topic 8: Software Development Process* to get more information about the different phases.



Lesson 1: Basic Explanation of DevOps

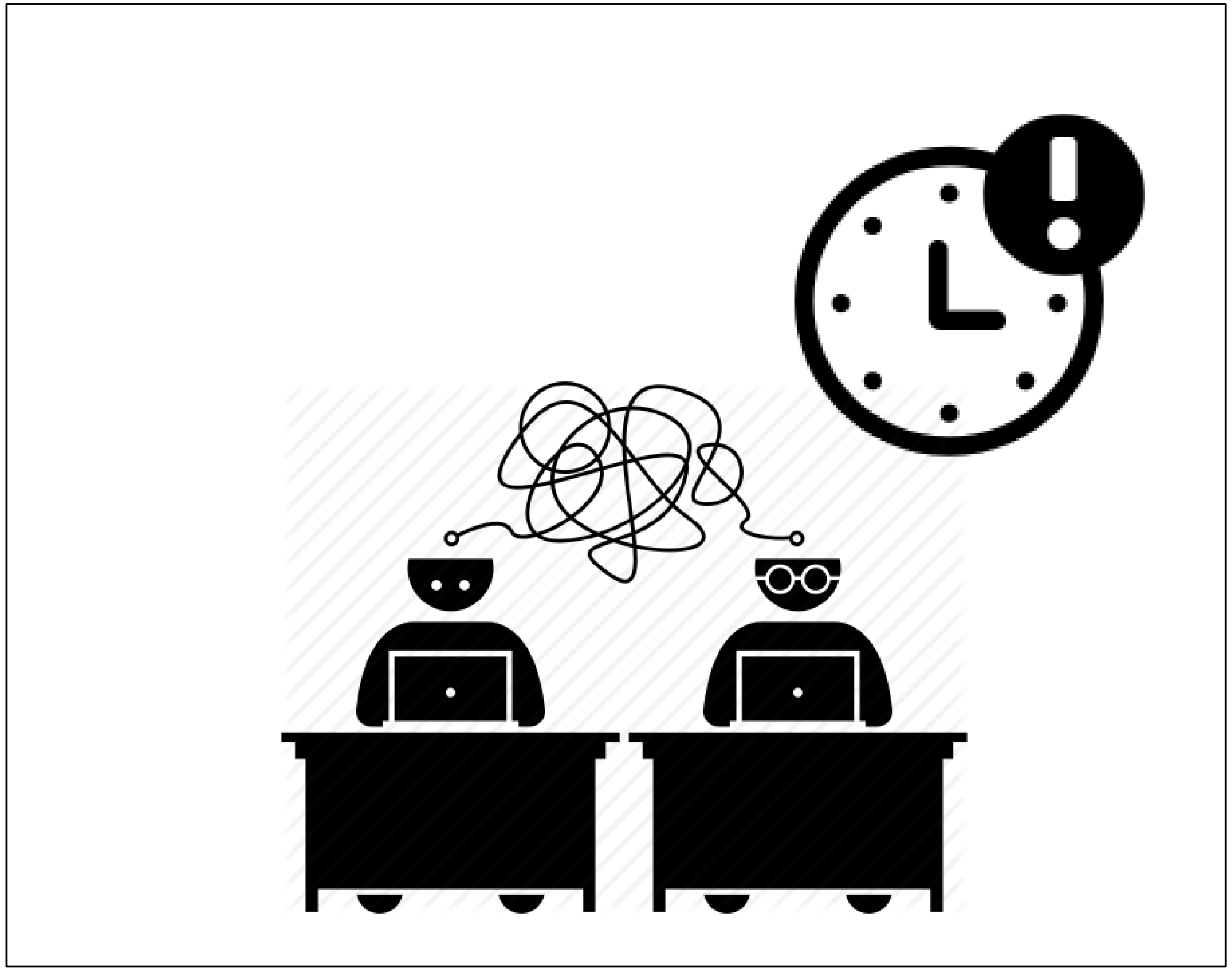
During the time of the waterfall methodology, *development* and *operations* were two different silos which were quite separated from each other. Development would worry about the creation of the software, throw it "over the wall" to operations, who had to maintain the software as long as it was alive (this almost sounds like giving away your child for adoption).

So, development creates a piece of software and gives it to operations to watch it and take care of it afterwards? That means that if something breaks in the software, operations has to fix it while actually development knows how it is built. That does not sound very logical, right?



Lesson 1: Basic Explanation of DevOps

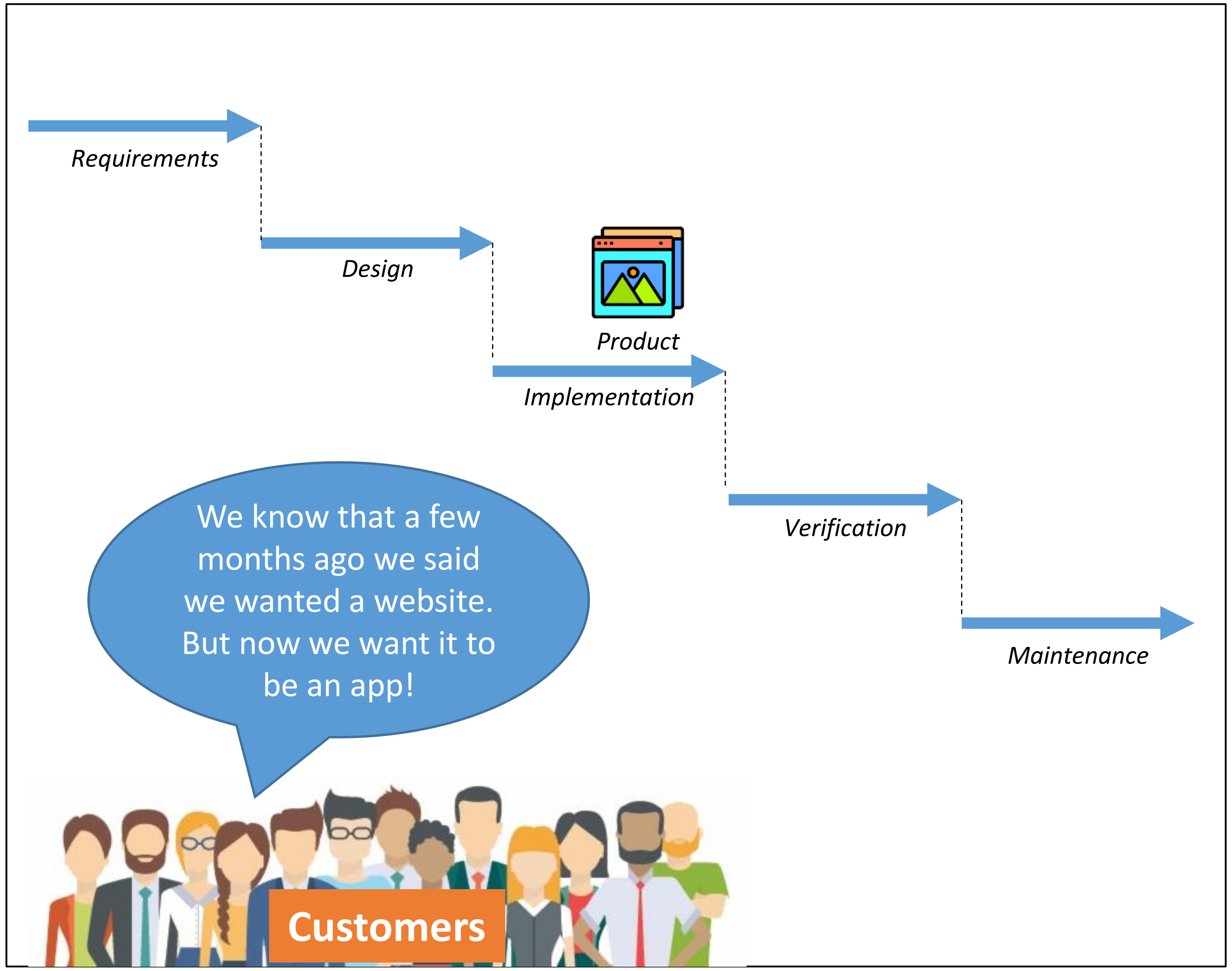
Besides the silos, the waterfall methodology caused some problems. Many projects were facing production delays, misalignment between teams, and poor communication between teams.



Lesson 1: Basic Explanation of DevOps

Production delays were mainly caused by the fact that a next phase could only start once the previous phase had been finished. This made the development of software very slow.

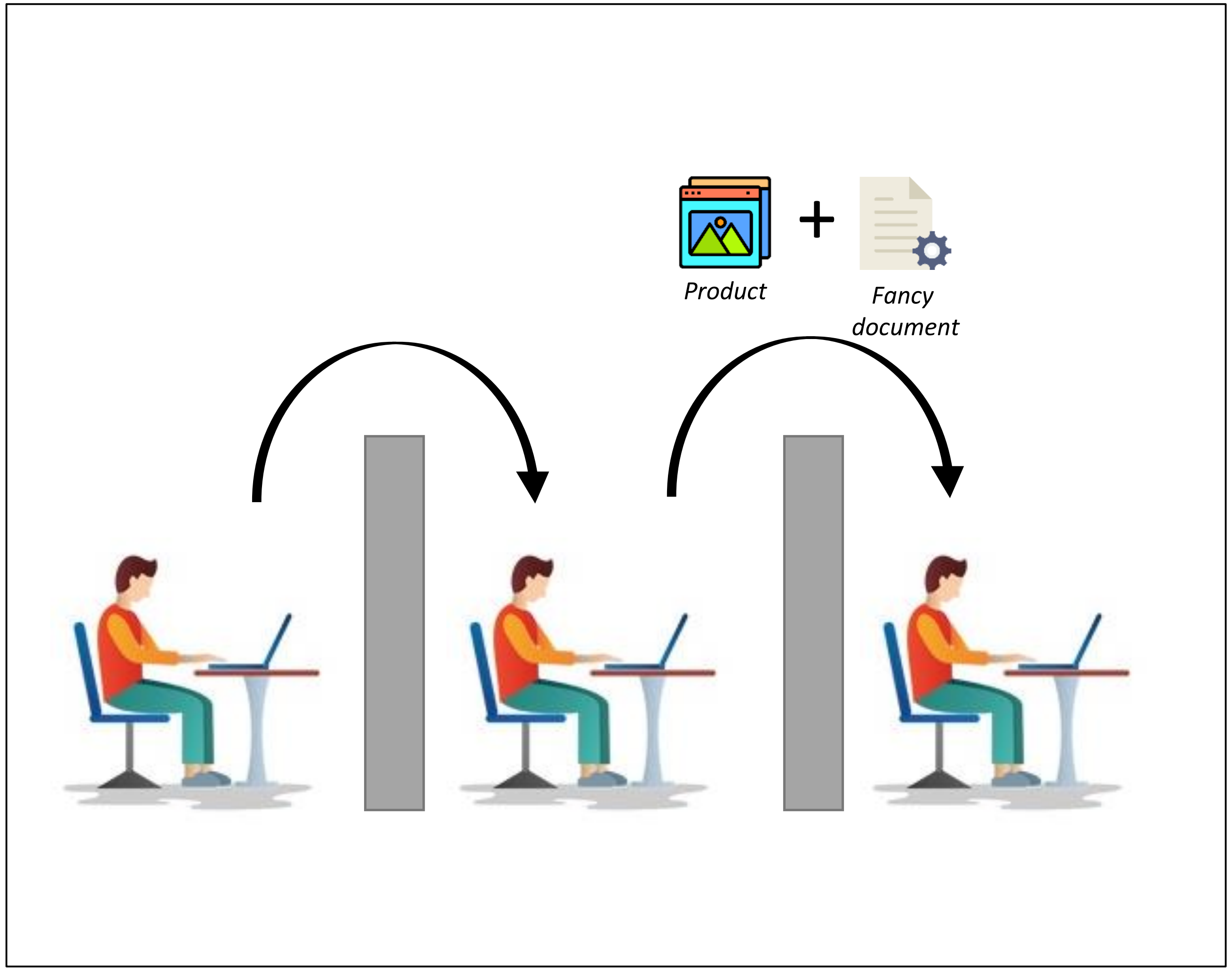
Now imagine what would happen if also the requirements or market demands change throughout the process. Total disaster.


Requirements
Design
Product
Implementation
Verification
Maintenance
We know that a few months ago we said we wanted a website. But now we want it to be an app!
Customers

Lesson 1: Basic Explanation of DevOps

Misalignment and poor communication between teams, including the silos of development and operations, were also a common problem. The culture was very much focused on doing your own task, and then throwing you work over the wall to the next person who had to work on it. Seems like people were not so aware of the bigger process and ecosystem that they were working in.

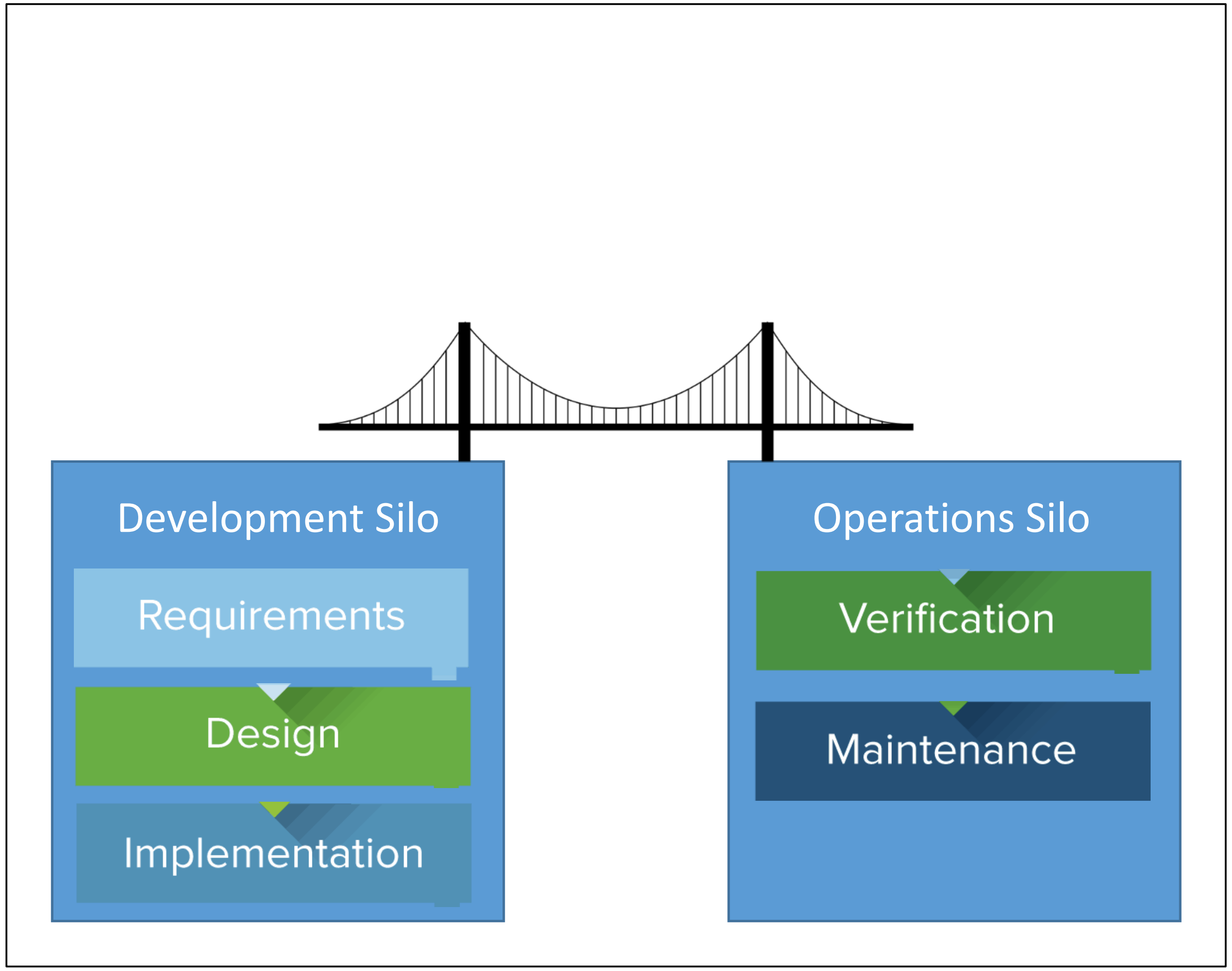
Communication did not extend much further than formal documents that marked the end of a phase. Of course, the next person to work on the project had to decipher that fancy document first in order to actually start working on the project.



Lesson 1: Basic Explanation of DevOps

Alright, enough whining about the past. Luckily, DevOps was introduced as a new way of working to prevent these problems. Please do note that at AFKL, we do not work entirely according to the DevOps approach.

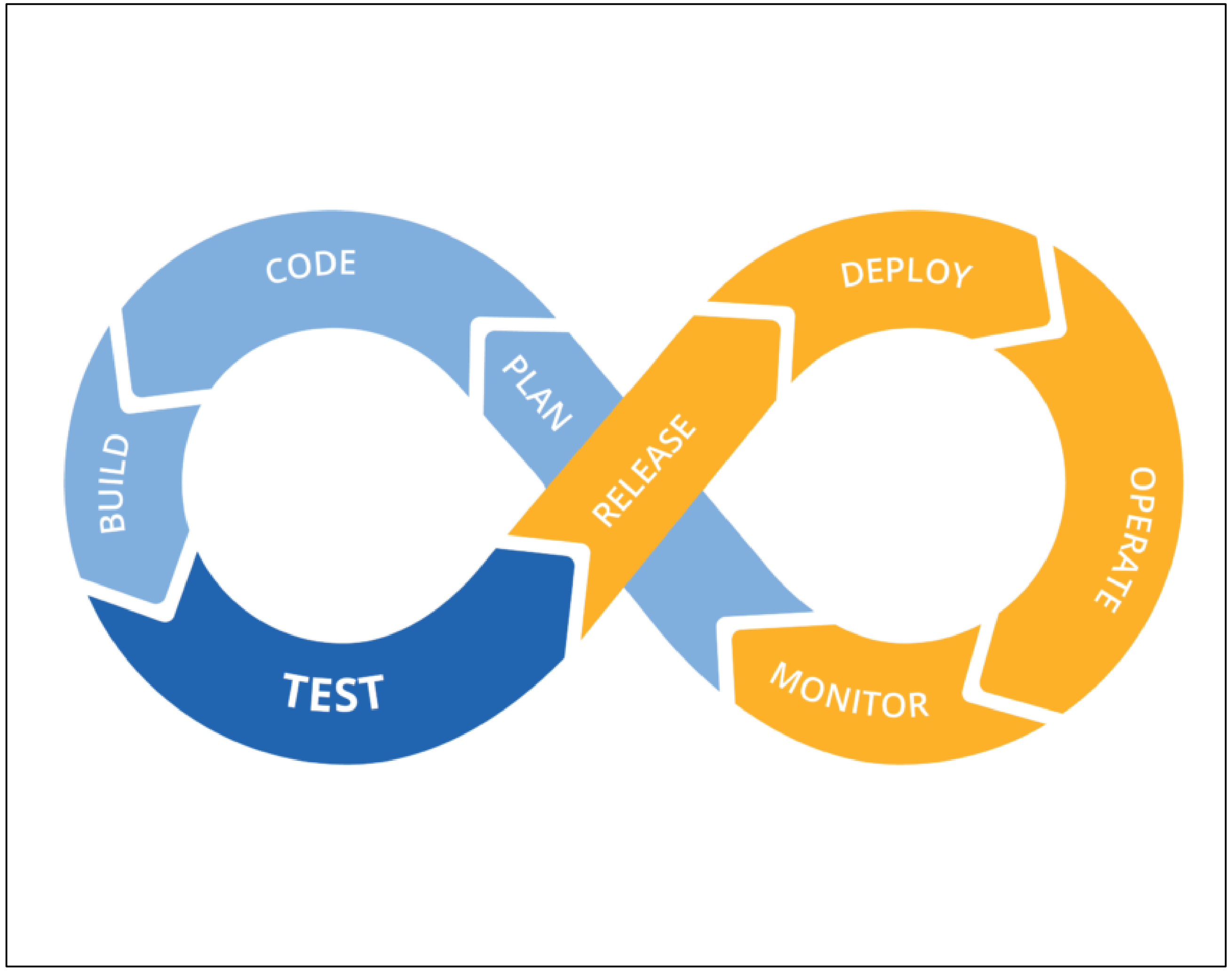
DevOps bridges the gap between development and operations, breaking down the different silos that we know from the past.



Lesson 1: Basic Explanation of DevOps

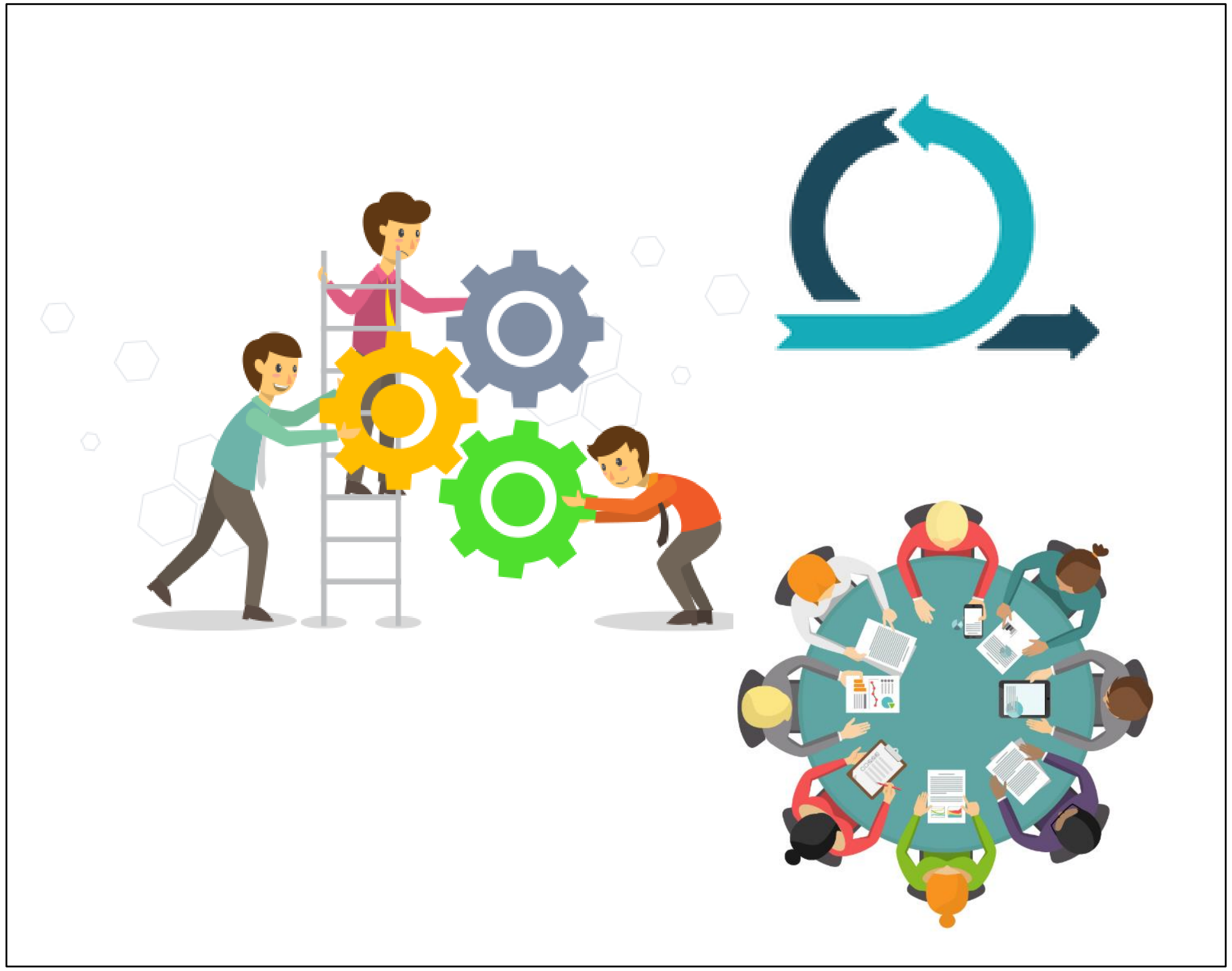
Practically the adoption of the DevOps way of working means that the software development team manages the entire application development lifecycle from beginning to end.

So no more throwing things over the wall and no more communication via fancy documents. Instead, the lifecycle phases will be addressed in short cycli and there is real-time communication between different teams. A huge step forward.



Lesson 1: Basic Explanation of DevOps

The DevOps way of working is focussed on agility, collaboration between different teams, automating the process as much as possible, and measuring performance. Sounds nice, right?



Lesson 2: DevOps Goals

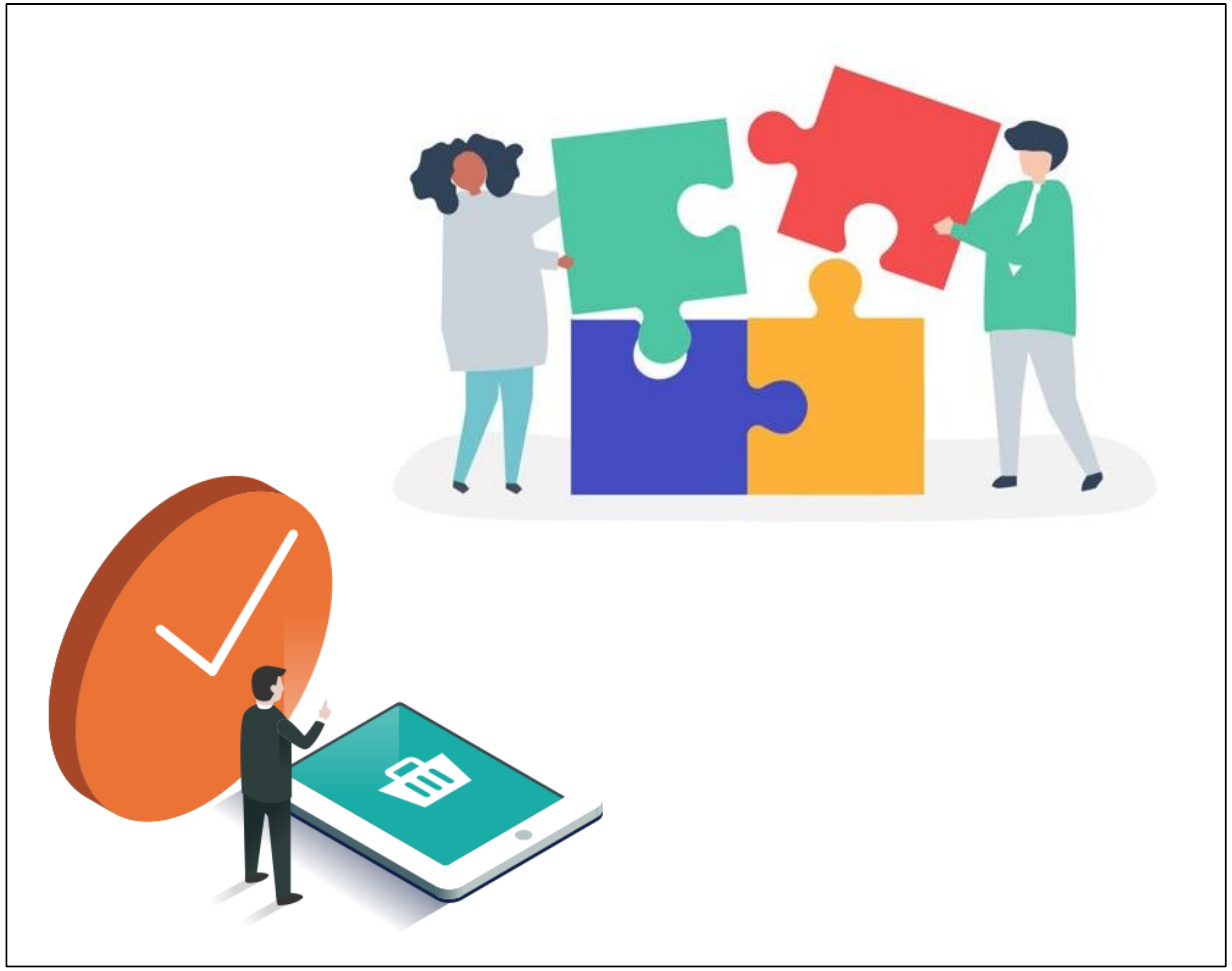
Lesson 2: DevOps Goals

So now that you roughly know what it means to work in a DevOps manner, let’s see what the underlying goals of this way of working are.

Lesson 2: DevOps Goals

The main goal behind the DevOps way of working is to optimize the flow of value from idea to end user. This basically means that the following sub-goals will be targeted:

- Faster time to market
- Lower failure rates of new releases, with faster mean time to recovery
- Better communication and collaboration between teams
- Addressing the (ever changing) needs of the customers



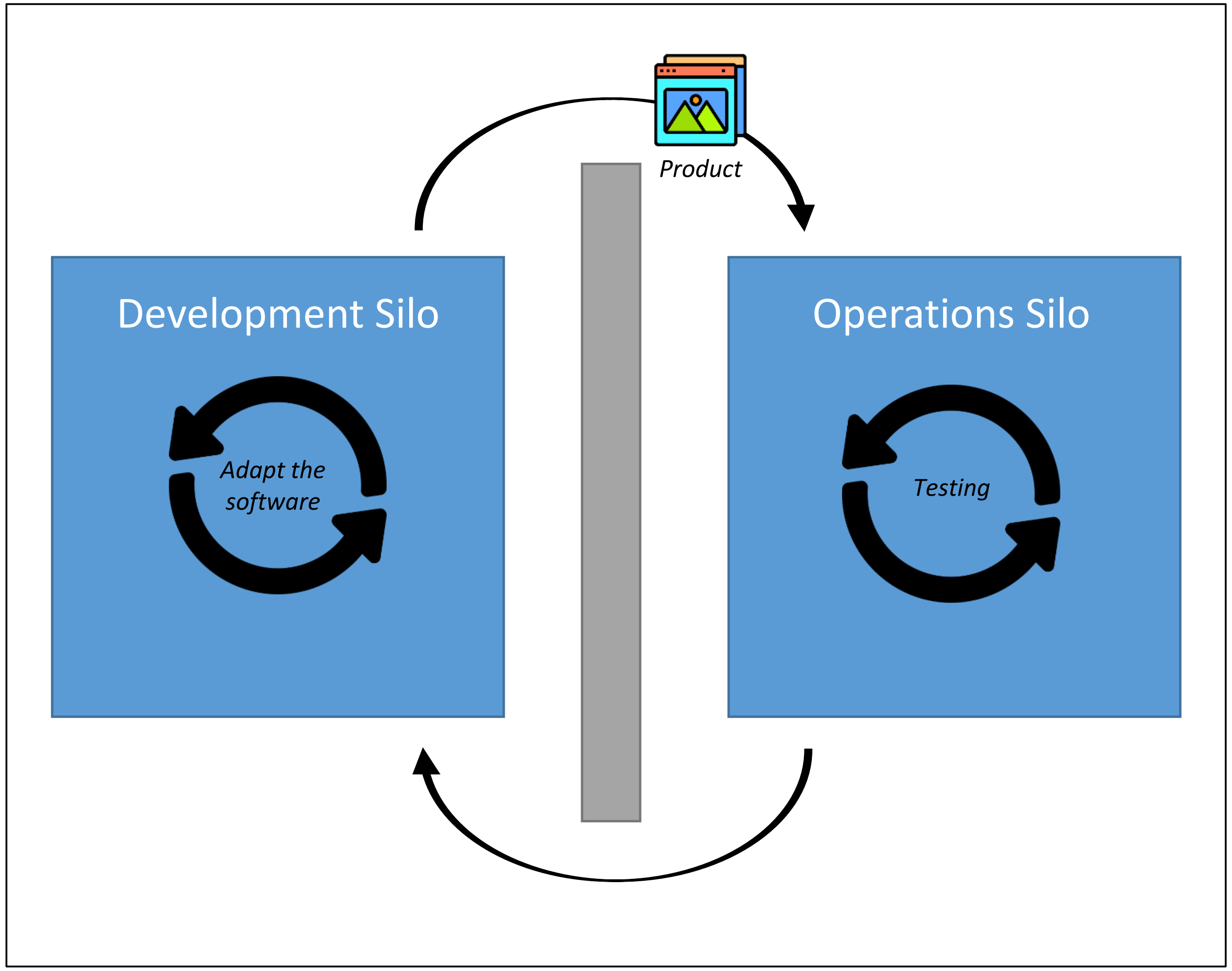
Lesson 2: DevOps Goals

Faster time to market is mostly reached by automation. For example automated tests.

In order to know if the software that you developed is any good, it is tested. Extensively tested, if you do it well.

Before the time of DevOps and automated tests, the software had to be sent to operations, who had to test it manually. This meant more work to do for operations and long waiting times for development.

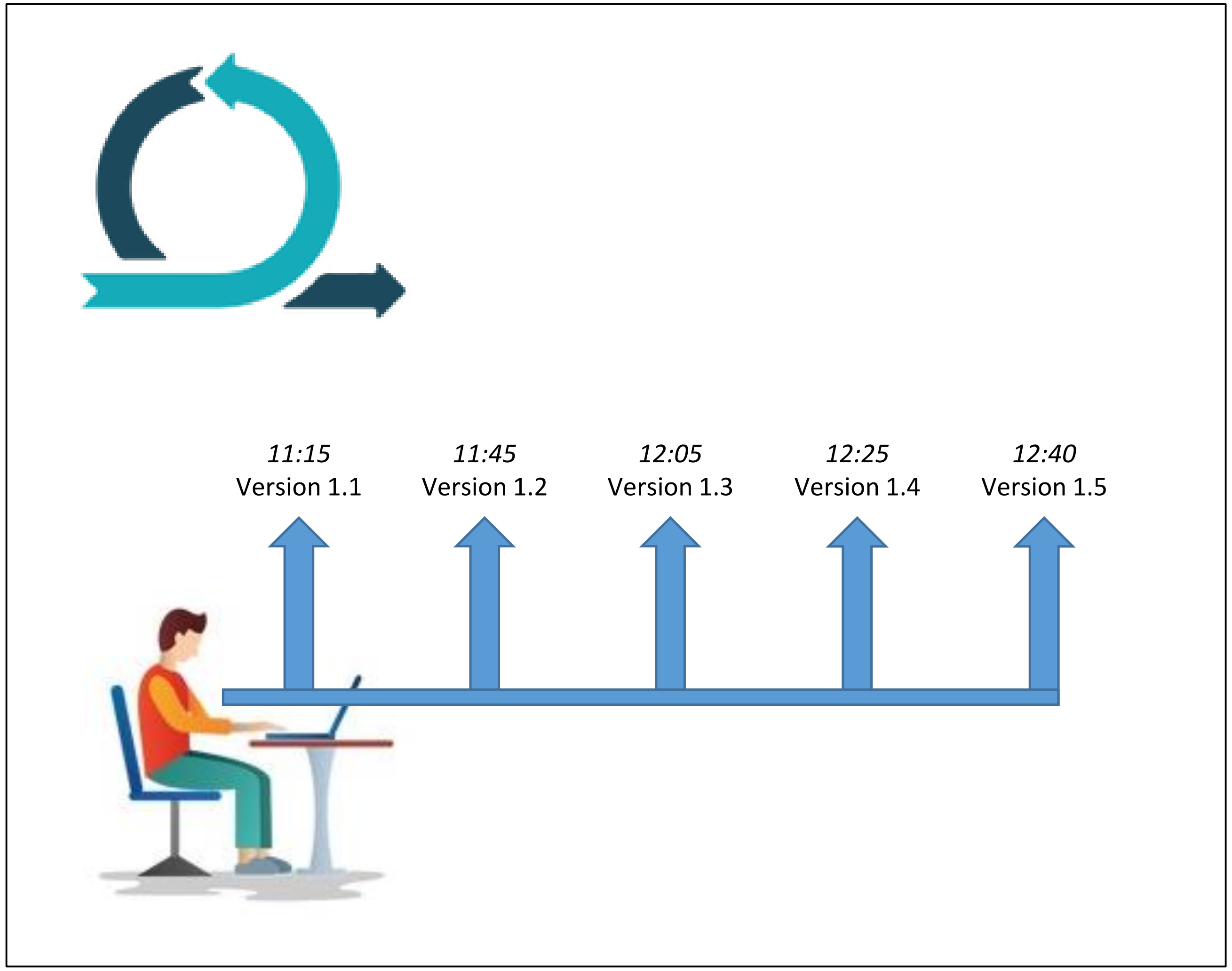
When integrating this in the DevOps way of working by having automated tests, the amount of work and waiting times decrease. This allows software developers to focus more on other tasks, which means faster software delivery.



Lesson 2: DevOps Goals

Lower failure rates of new releases and faster mean time to recovery are reached by agile (programming) practices introduced by DevOps.

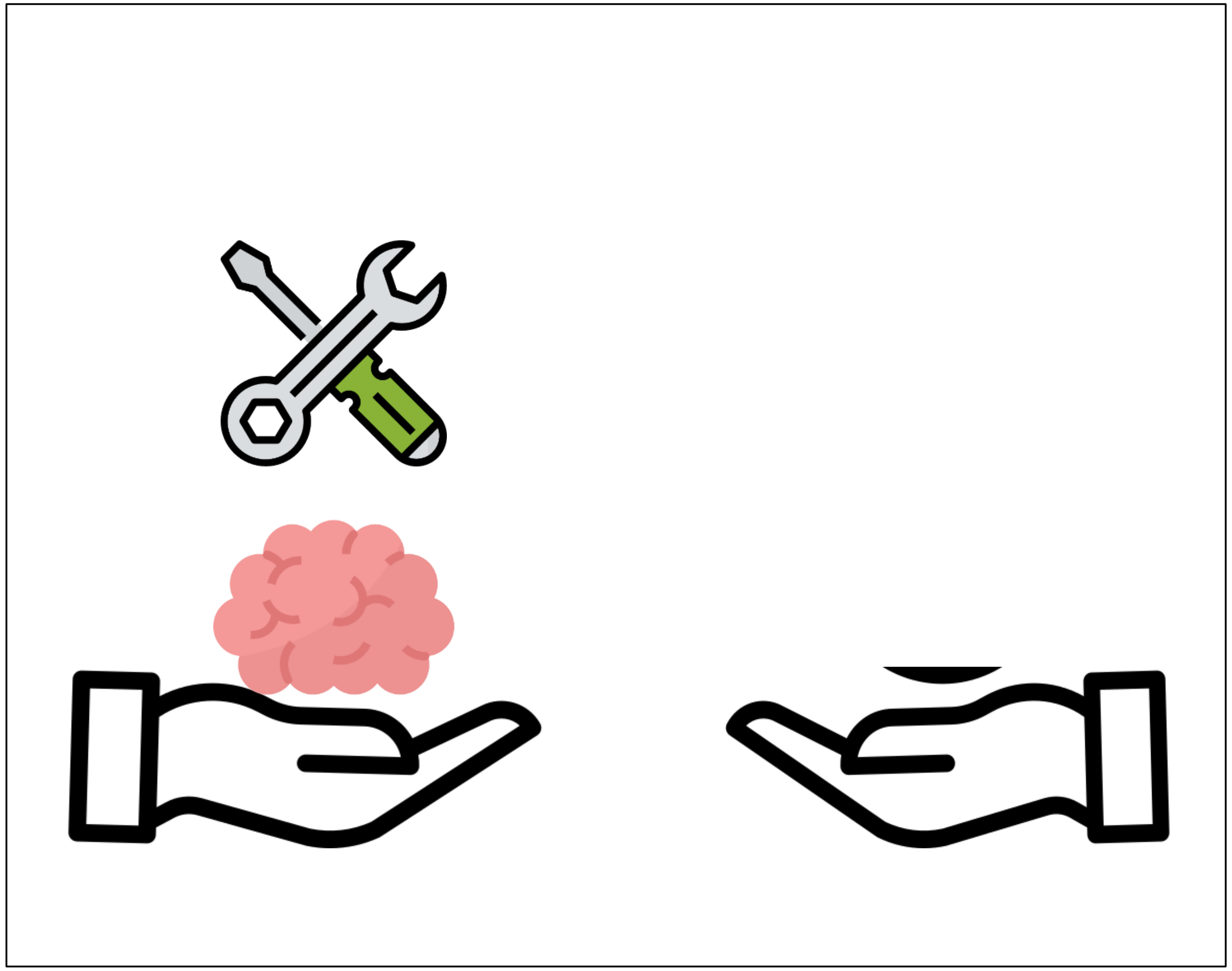
Short development cycles with frequent code versions and revisions help to track when and where the software breaks down due to bad code. If a failure is detected, the team can easily roll back to an earlier version of the code which had already proven to work well. Furthermore, the small changes between different code versions make that it is easier to recover from problems, as only a small part of the code causes the problem and has to be changed.



Lesson 2: DevOps Goals

Better communication and collaboration between teams is another important goal of DevOps. Remember that DevOps tries to bridge the gap between development and operations? Well, how are you going to do that without any communication and collaboration between teams?

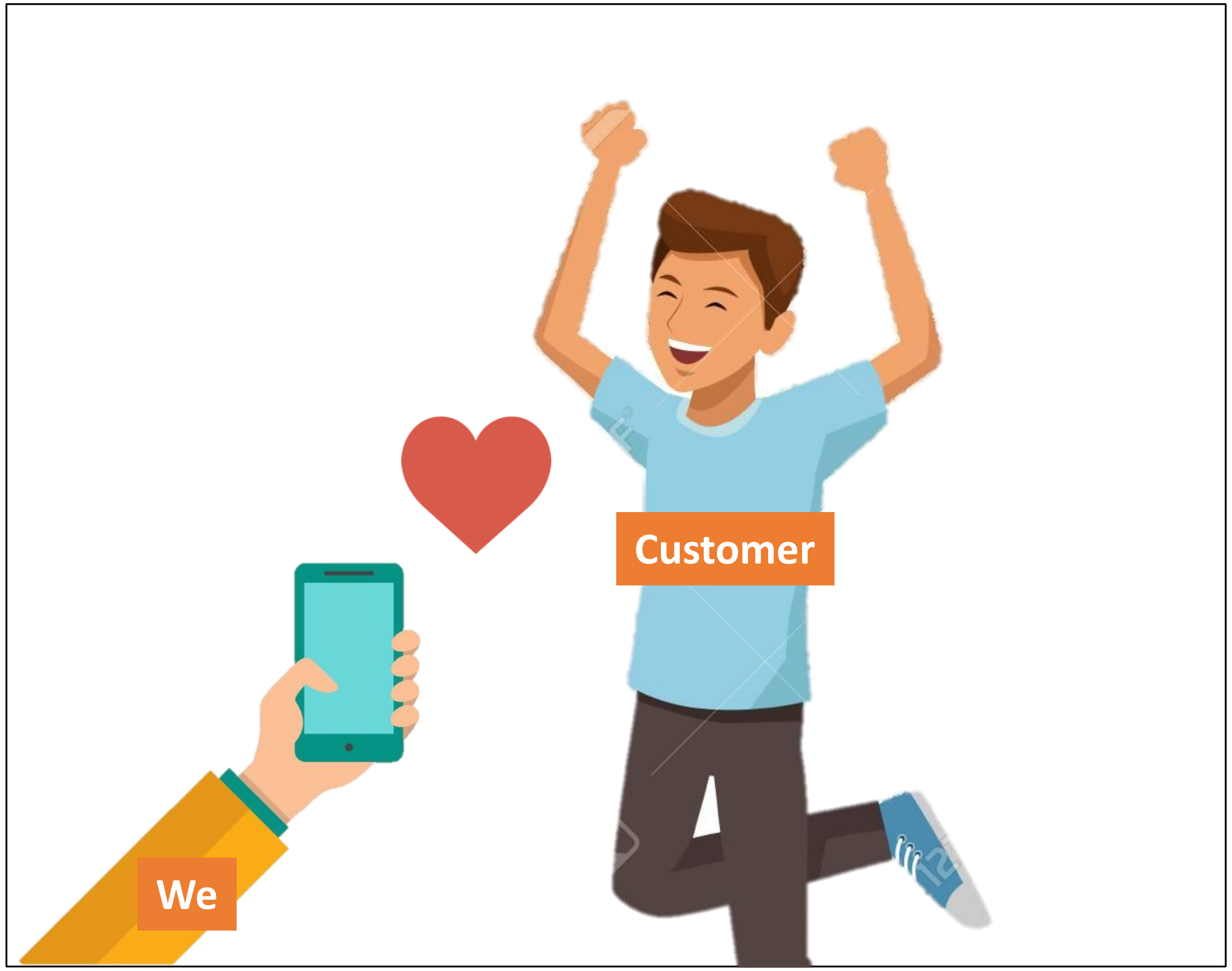
The key is to be transparent about the work and to share knowledge, experiences and tools with one another. Why let another team reinvent the wheel if you know how it's done? This approach might sound logical, but if we think back to the old waterfall days with siloes and fancy documents, we see that it has not always been like this.



Lesson 2: DevOps Goals

Finally, DevOps is all about being aware of the end-user. Therefore, we aim to address real customer needs, even though these might change quite frequently. This way, we try to delivery real value to the customer, something they really want to have.

How do we do that? Well, we do something called business monitoring. You will be able to find out more about that under *Lesson 10: Monitor*.

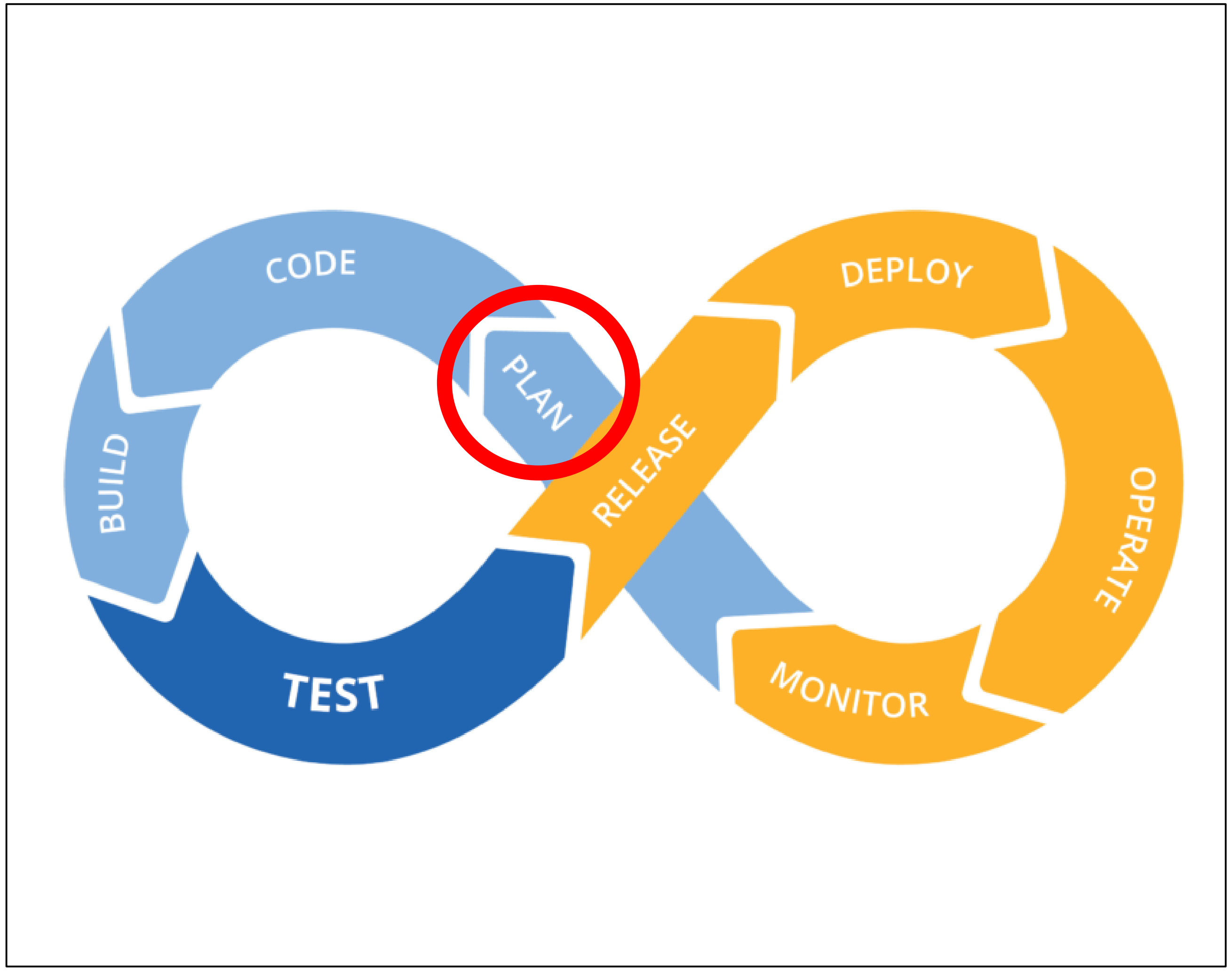


Lesson 3: Plan

Lesson 3: Plan

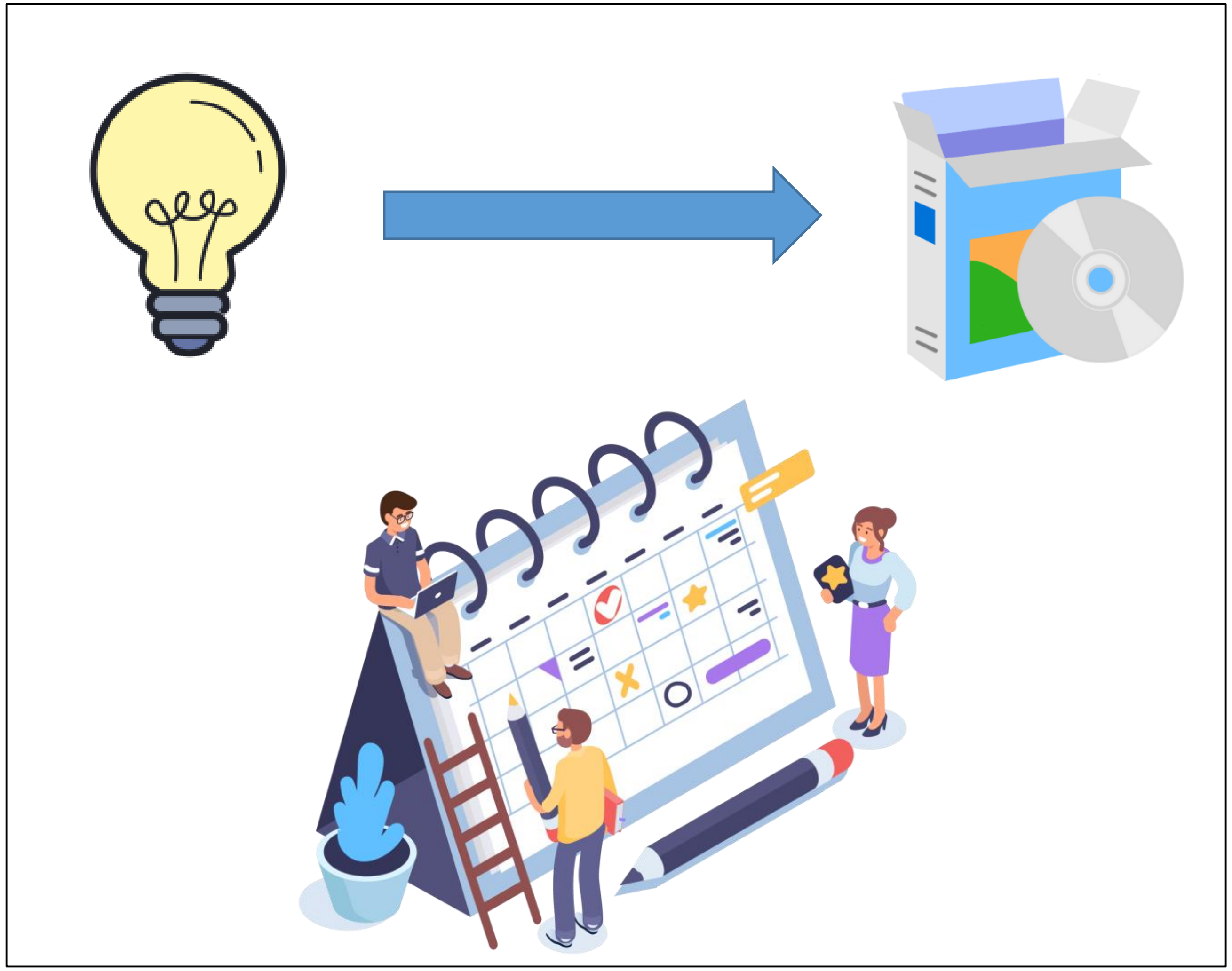
Remember the application development lifecycle which is now entirely managed by the software development team? Let’s use the upcoming lessons to go through the different phases of this lifecycle to see what actually happens when working in a DevOps way. To begin with the planning phase.

Before we start with this, please note that different phases can happen simultaneously and might be revisited. Therefore, the order in which they are presented is just a guideline, not the ground truth. Let’s keep it agile, right?



Lesson 3: Plan

As the name already suggests, during this phase the planning for the upcoming period will be made. All the steps required to go from a service or product idea to a realization that can be given to the customer are to be included in the planning.



Lesson 3: Plan

The development team sits together with the stakeholders, who represent the customer. Together, they determine what the business case is and what will be offered to the customer at the end of the cycle.

The stakeholders will determine (non) functional requirements of the solution and will allocate budget to it. You can find out more about this if you go to *Lesson XX: Requirements Analysis.*

Lesson 3: Plan

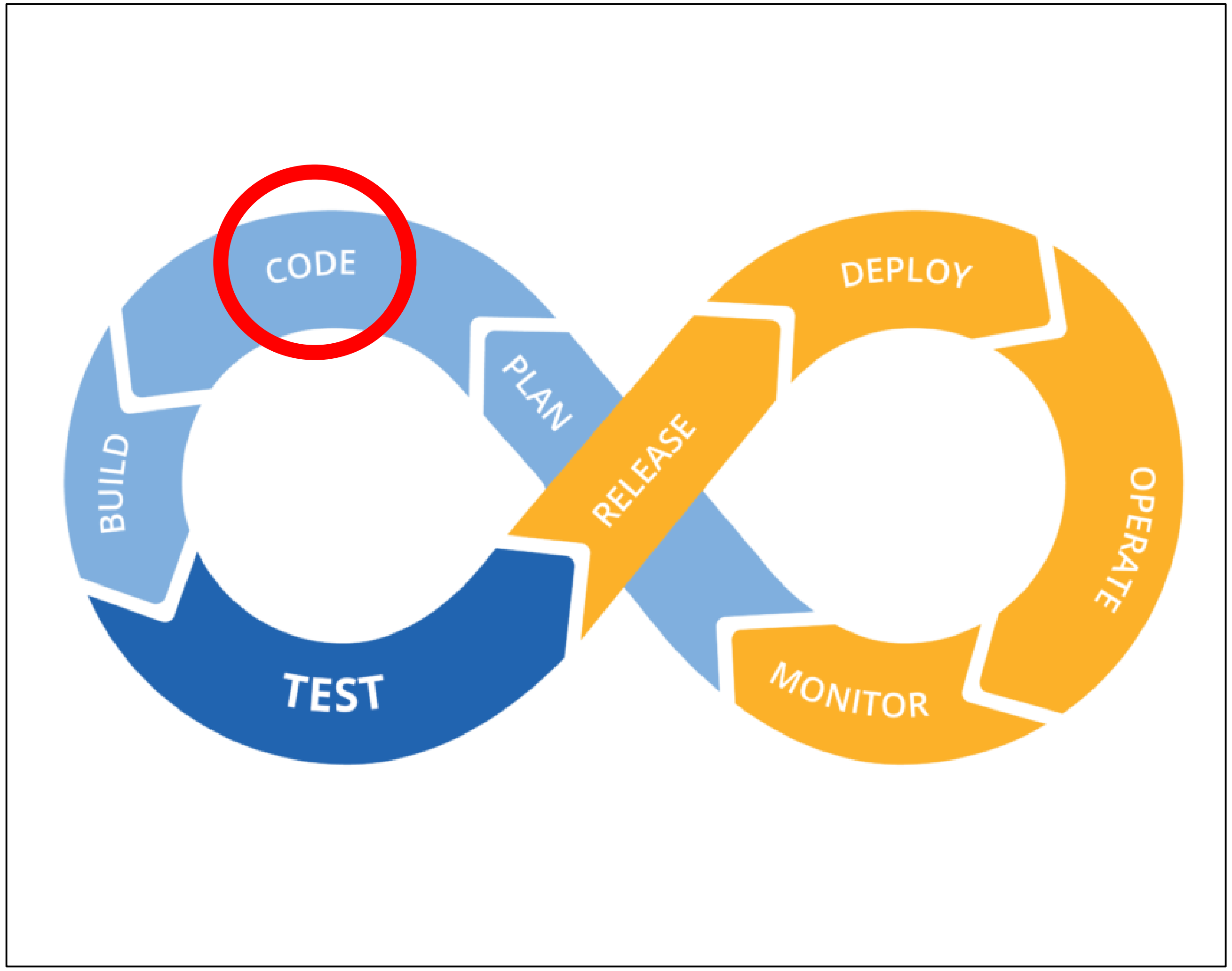
When it is clear what the business case is and what will be offered to the customer, the roadmap and approach are determined. The roadmap basically contains the steps that will be executed to get to the final solution, usually with certain milestones along the way. The approach defines how the team will go through the cycle. Within AFKL Digital, Scrum is always the approach. See *Topic 1: Scrum* to learn more about this.

By the end of this phase, the software development team should know what is expected from them and how to start working.

Lesson 4: Code

Lesson 4: Code

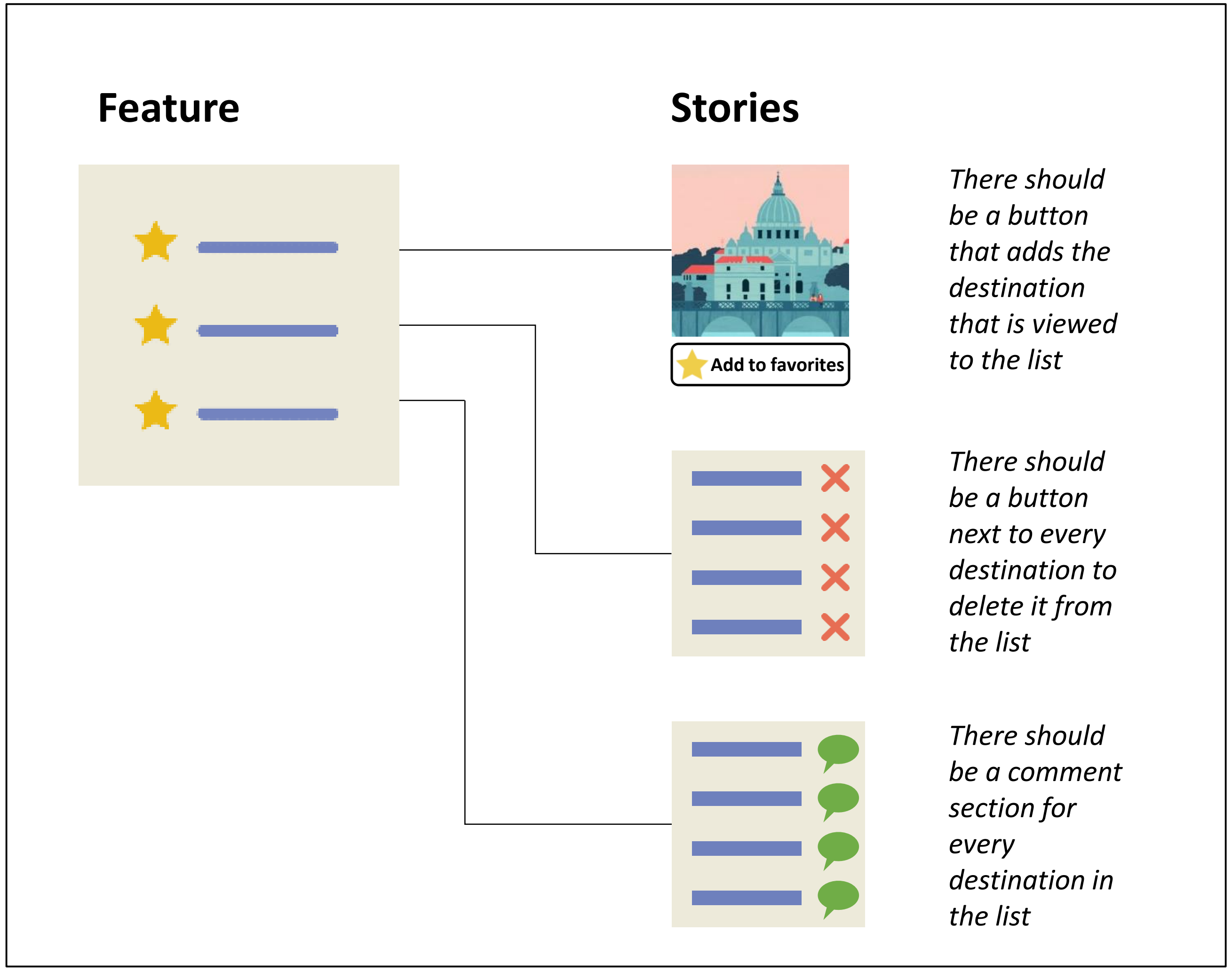
The next DevOps phase is the coding phase. During this phase, the software is designed and created. For the developers this basically means that they start to write code.



Lesson 4: Code

Of course, one cannot start writing code out of nowhere (at least, not if it should be useful code). Typically the step from the planning phase to actually writing code is too big.

What does result from the planning phase is a list of features. Features are basically descriptions of what should be delivered by the end of the cycle in business terms. An example of a feature for the KLM website could be a list of favorite travel destinations. A feature is translated into multiple stories, which tell the programmers what they should do in order to implement that feature. A story could for example say that there should be a button that adds the destination that is being viewed by the user to his/her list of favorite travel destinations.

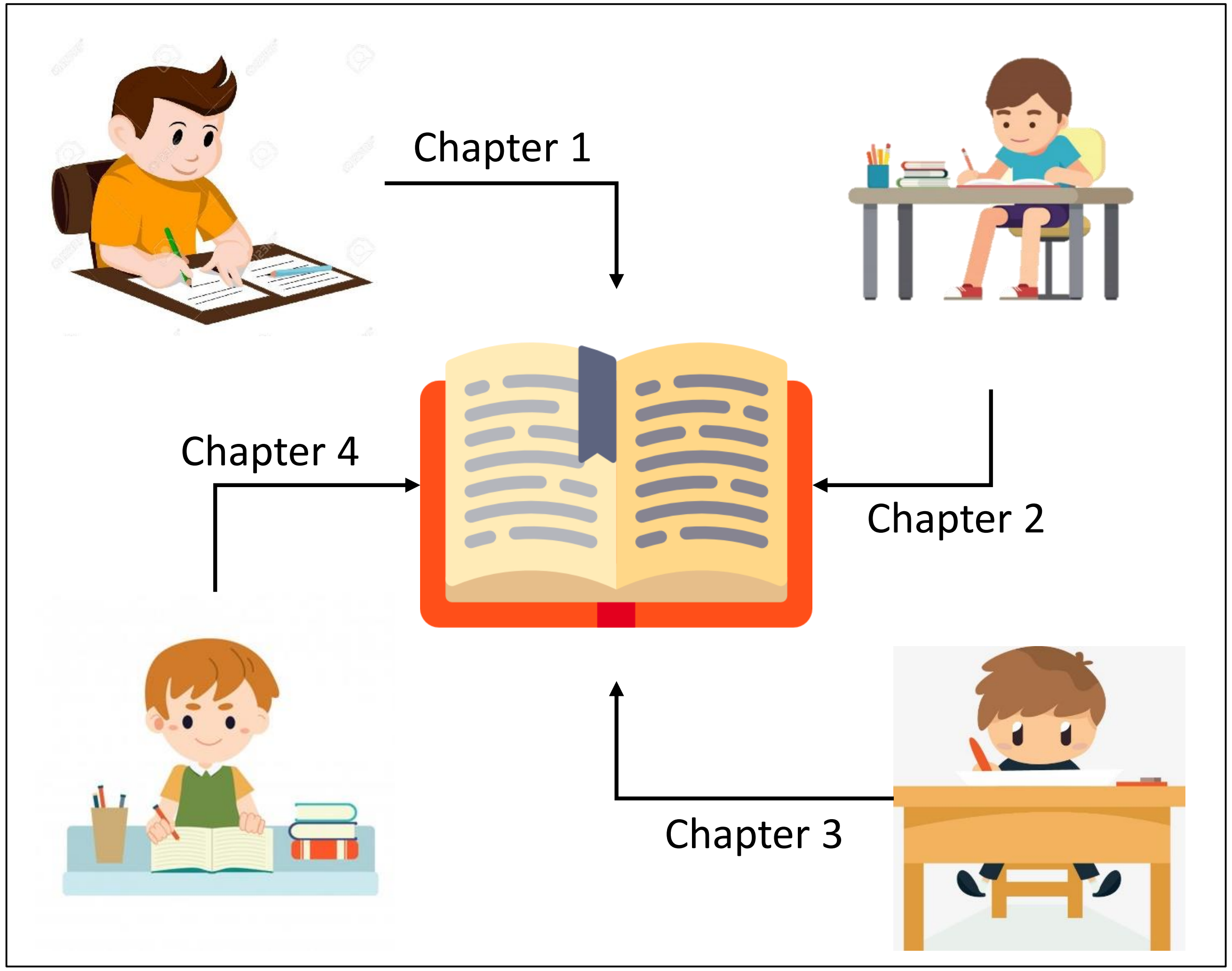

Feature
Stories
Add to favorites
There should be a button that adds the destination that is viewed to the list
There should be a button next to every destination to delete it from the list
There should be a comment section for every destination in the list

Lesson 4: Code

Okay, so these programmers start to write their code. But they usually do this as a group right? How do you write code together? Divide the work, maybe? But how do you then make sure that all the code fits together in one big application that works?

You can compare it to writing a book, right? If everyone writes a chapter, how do we make sure that the end product is a coherent story?

Version control, my friend. It is the answer to all your questions.



Lesson 4: Code

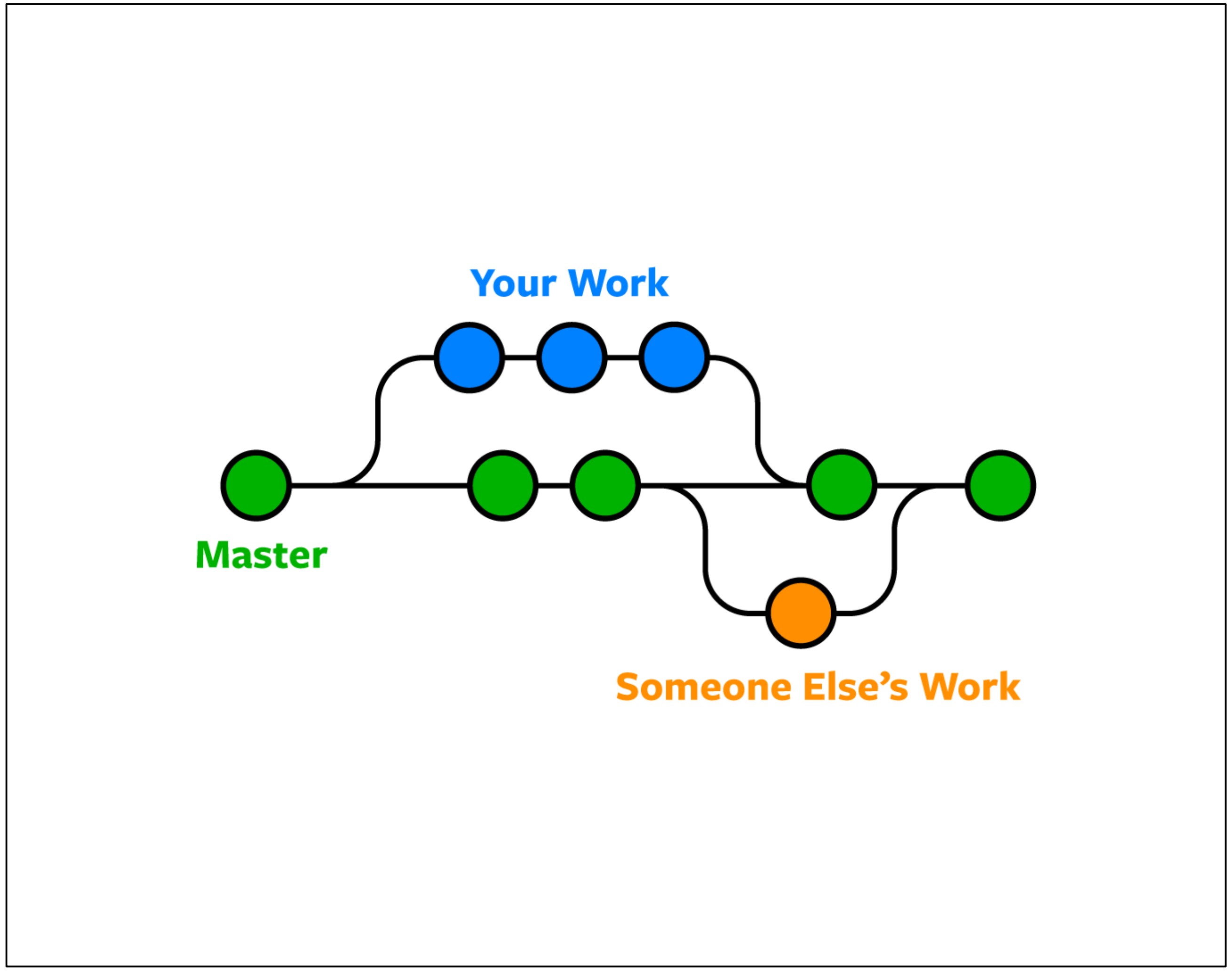
When coding, version control is extremely important. Remember that one of the DevOps goals is to have lower failure rates of new releases and faster mean time to recovery? This is realized by constantly keeping track of different versions of the code.

Typically, there is one ‘master code’, which is the code that works perfectly. It is so perfect, you would almost be afraid to touch it. Therefore, this code is not touched unless we are completely sure that the addition or change works perfectly as well.

Lesson 4: Code

Every developer works on a smaller part of the master code. They usually call this a 'branch'. Whenever a developer makes a change to his/her part of the code, this is uploaded to the version control system ('git' as most developers call it) and tested (automatically of course, remember?).

If the code change works perfectly, it will be added to the master code. If the code change does not work perfectly, it is of course not added to the master code and the programmer responsible for it has to get back to work to make it work perfectly for the next time. Of course, fellow programmers might help him/her in this case! Since everyone works together.

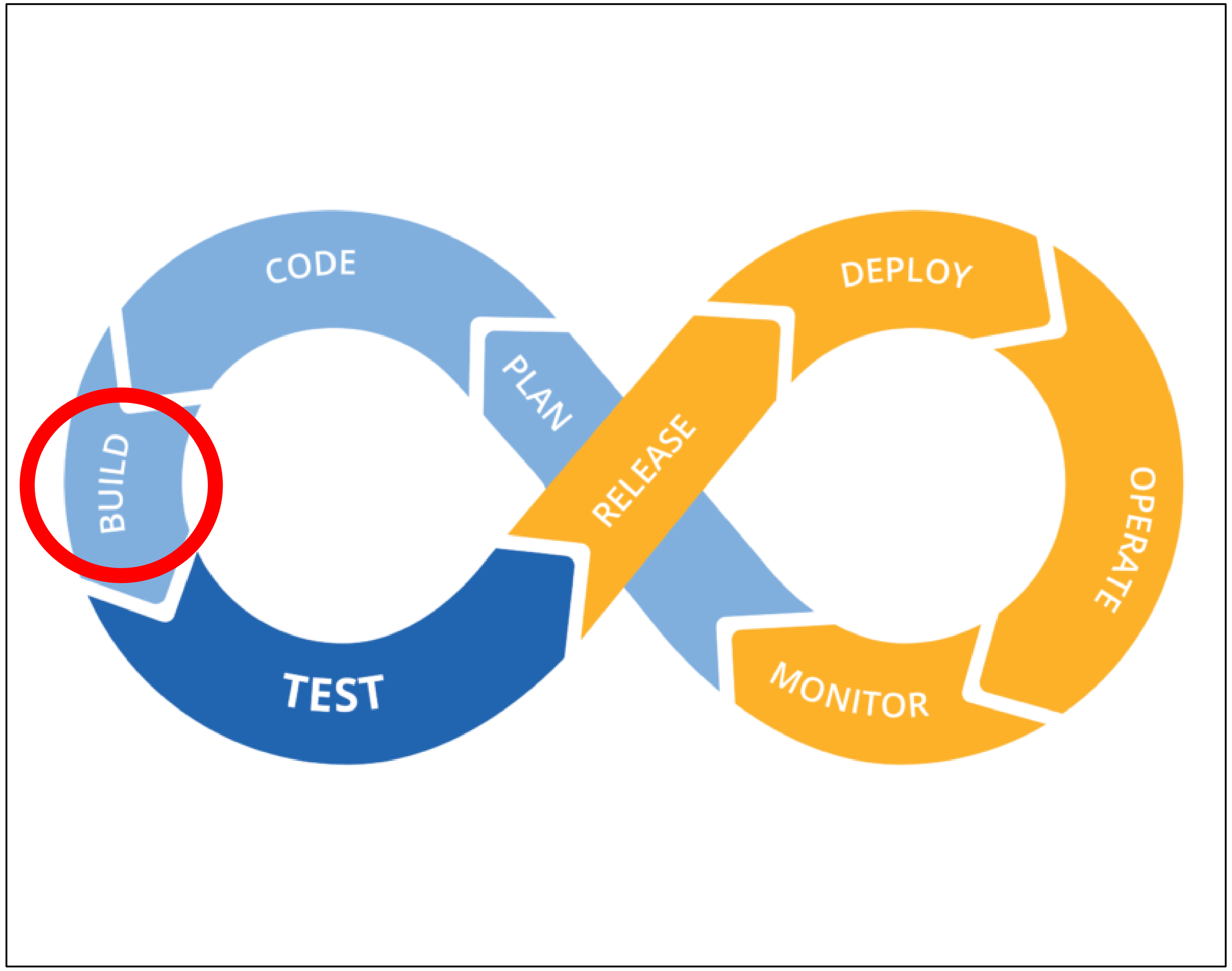


Lesson 5: Build

Lesson 5: Build

Once the code has been written, the developers should make a build out of it. Making a build is also called ‘compiling’ the code.

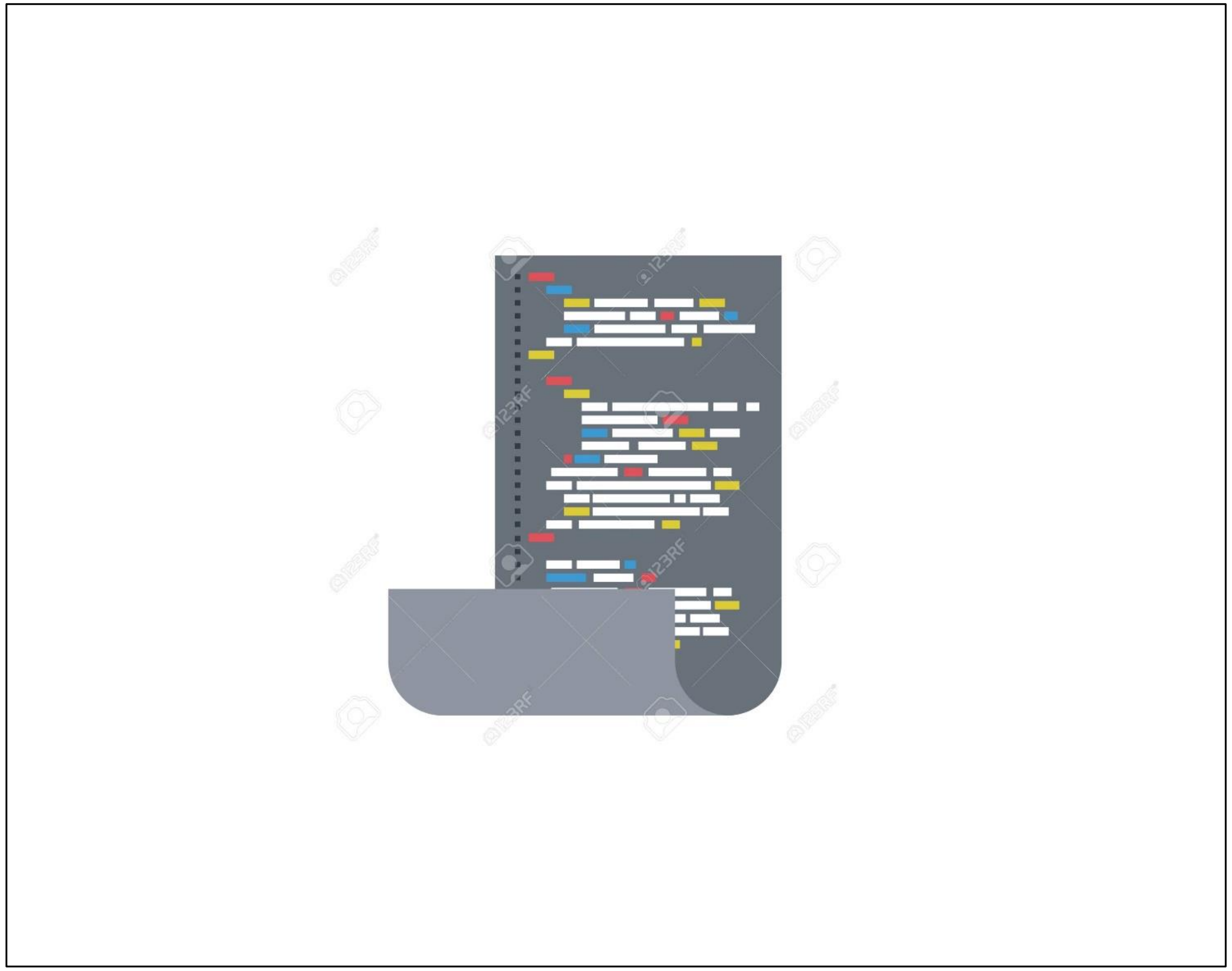
Making a build basically means that the code is converted from source code to object code.



Lesson 5: Build

Source code is the code as it was written by the programmer(s). For example, a nice piece of PHP. Unfortunately, this nice piece of PHP cannot run as an application on every device. It will only run if you have an IDE, integrated development environment, installed on your device and open the code through there.

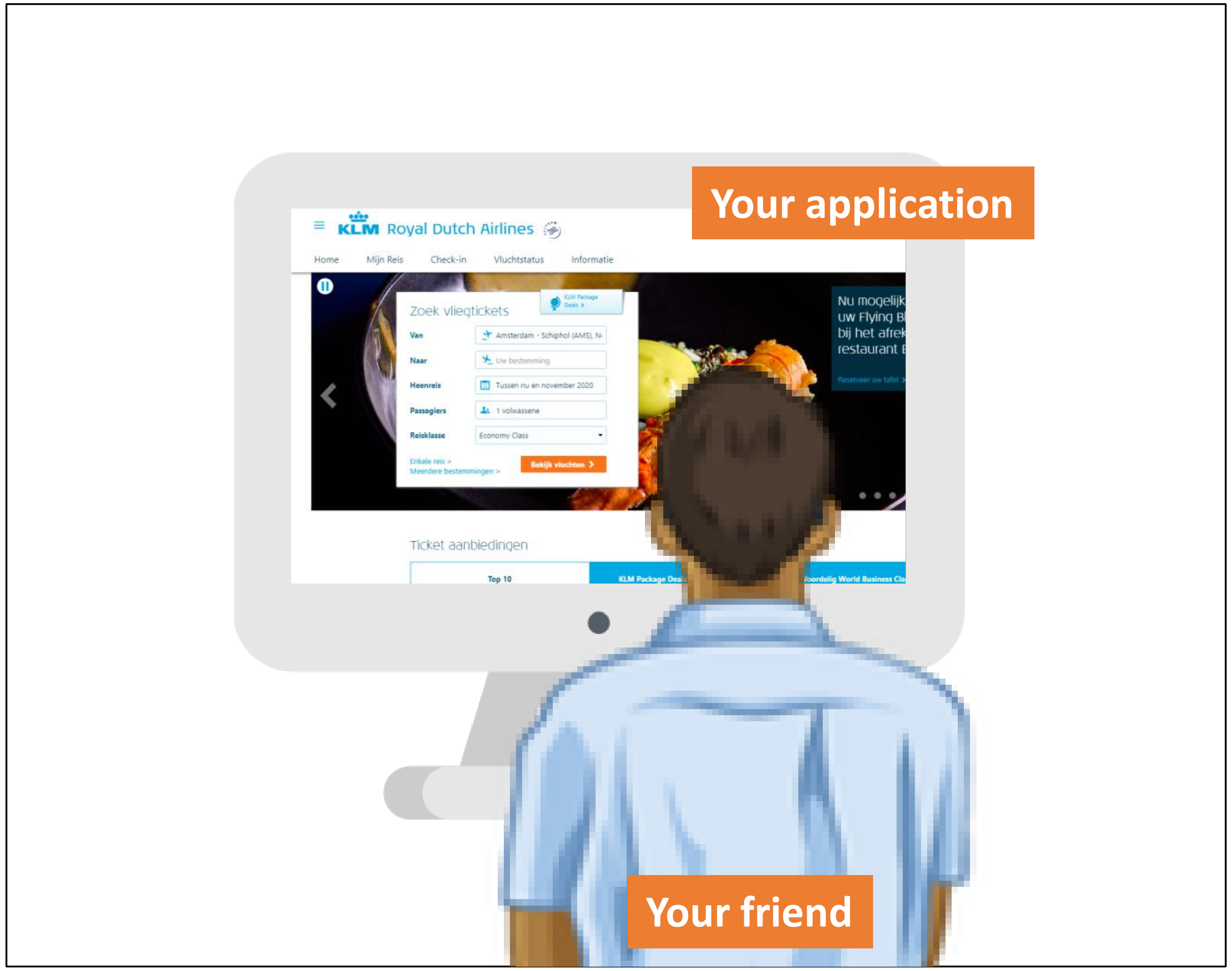
So, sadly, if you send your nice PHP code to your friends and they don't have the right IDE installed, they will just see lines of code instead of your beatiful application. Too bad…



Lesson 5: Build

This is why we make a build and convert the source code to object code. Object code is basically a package that can be read by an operating system (Windows, Mac or Linux). You could call it an application. Something stand-alone that a computer can run. This means that if you make a build of your code for Windows, you can send it to all your friends who have a Windows PC and they will be able to open and run it without having the IDE installed. Yay, finally people see your beautiful application!

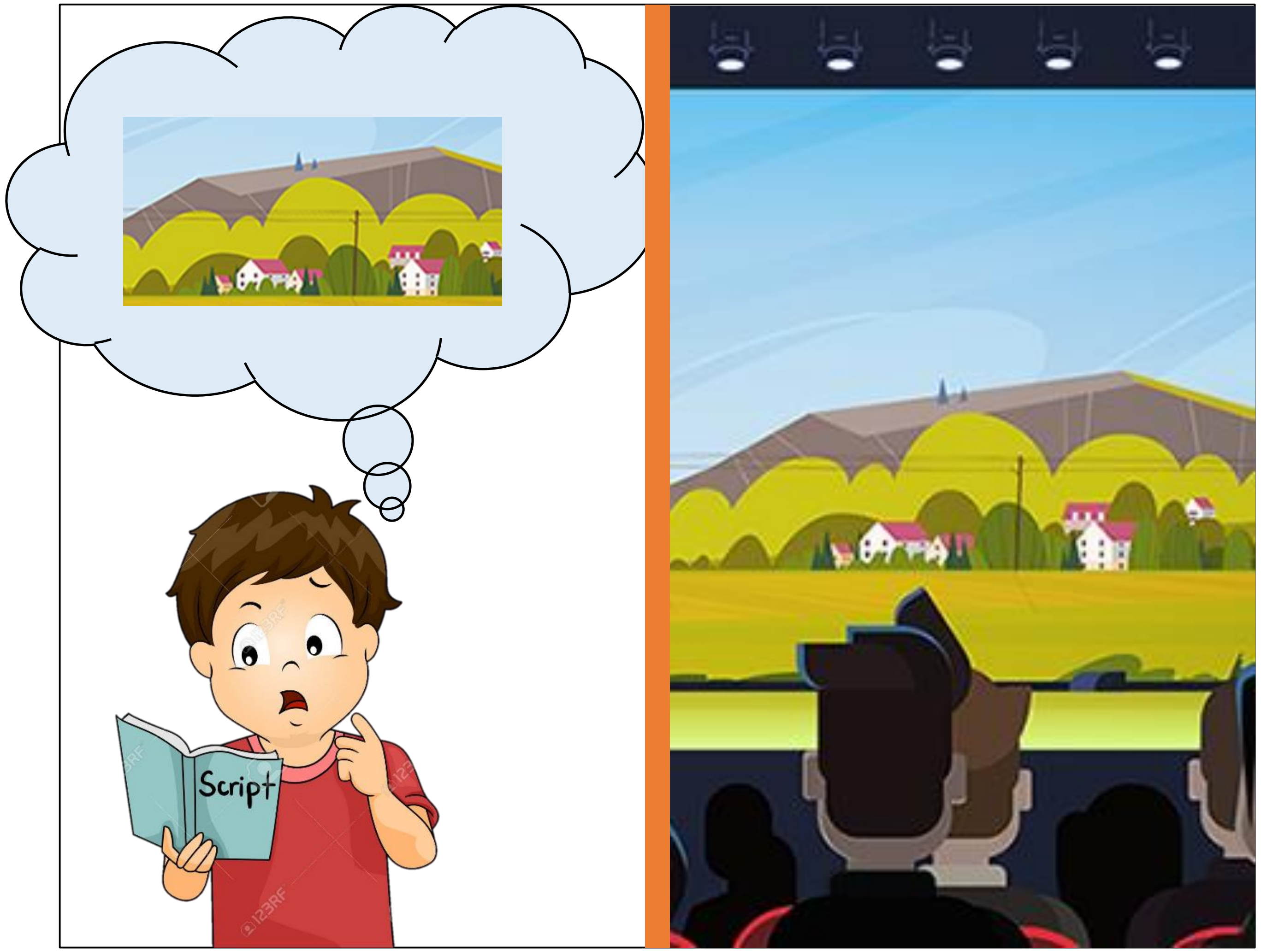
Unfortunately, your Mac friends will be left out… Since Windows builds don't work on Mac and vice versa (try downloading software from another operating system to your PC, you won't be able to install it).



Lesson 5: Build

You could would compare the process of making a build to the creation of a movie. At first, there is only the script, which is the source code. When opening the script, it is just a bunch of lines of text. You can see what will happen in the story, but you have to imagine it yourself.

Once the movie is produced, a build is made of the script, which turns it into object code. We can display it on the screen and see the movie running in front of our eyes. Much more user friendly than reading through an entire script and having to imagine what it would look like, right?

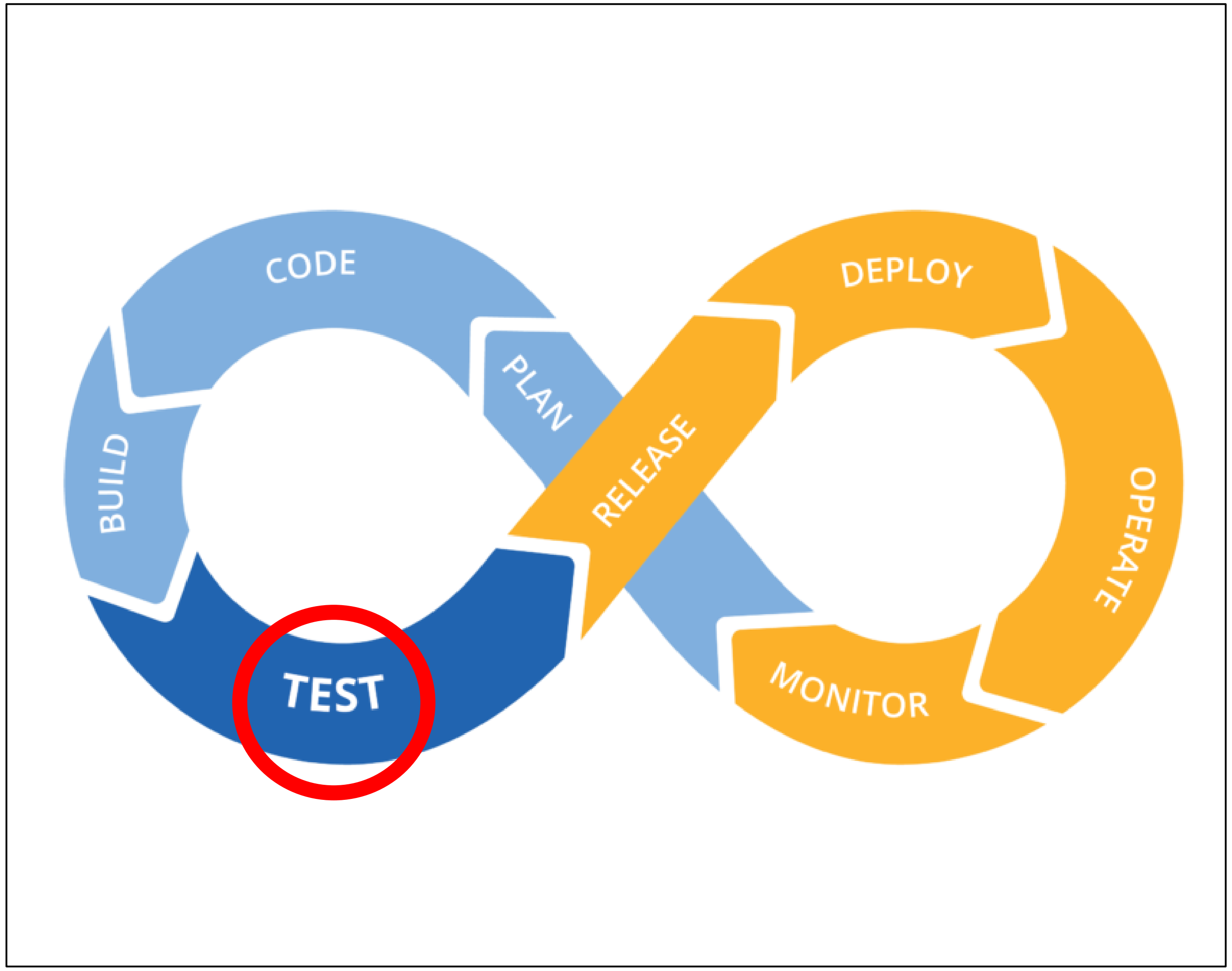


Lesson 6: Test

Lesson 6: Test

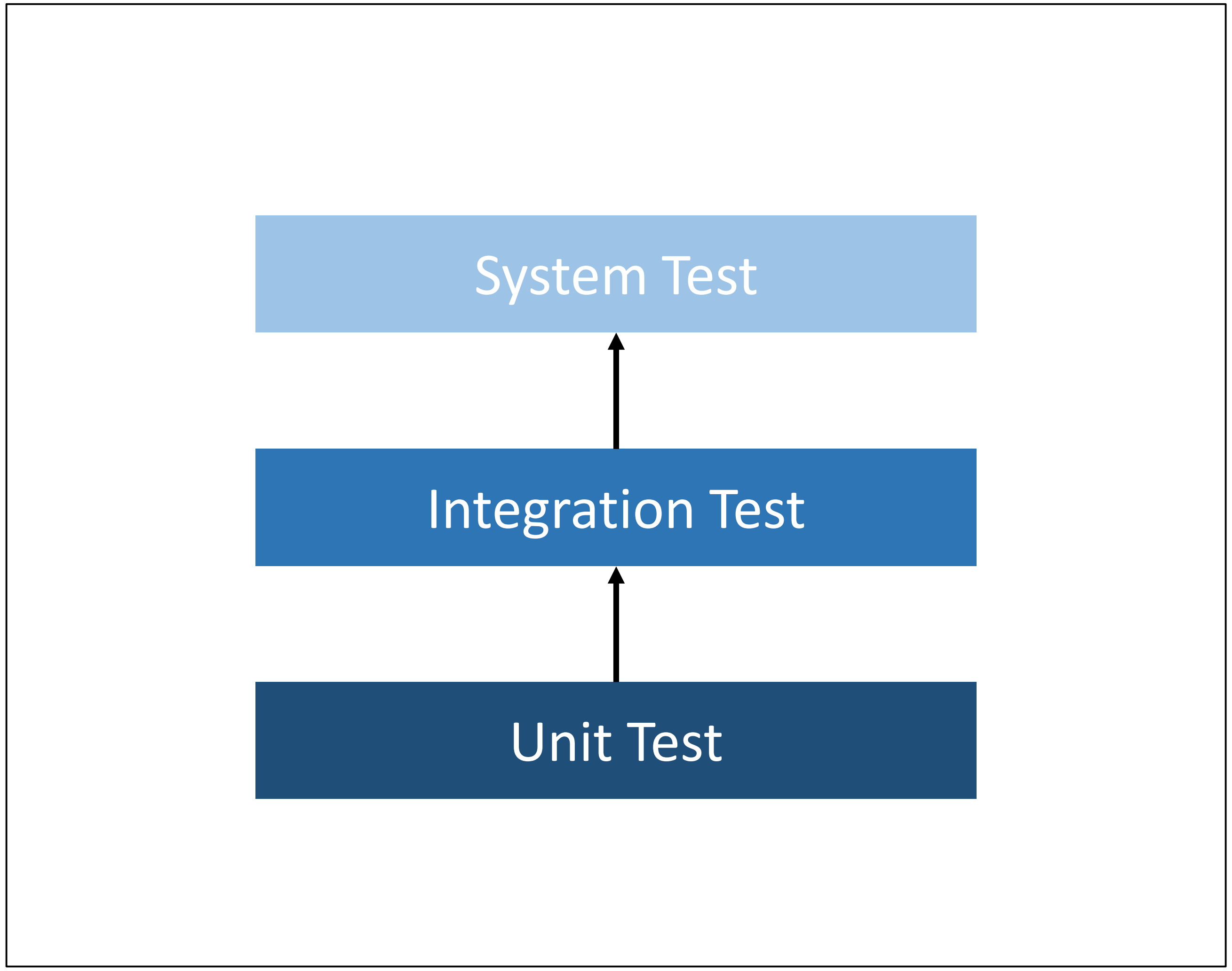
Of course, the main goal of the test phase is to test the code. Tests can happen manually or automatically. As you may expect by now, we prefer to do it automatically as much as possible.

It is important to test your code thoroughly. Why? Well, imagine what would happen if you give you application to the consumer and during use it suddenly stops functioning properly? Or even worse, what if the application causes serious security threats? This will for sure cause some angry or disappointed users. So let's try to prevent that.



Lesson 6: Test

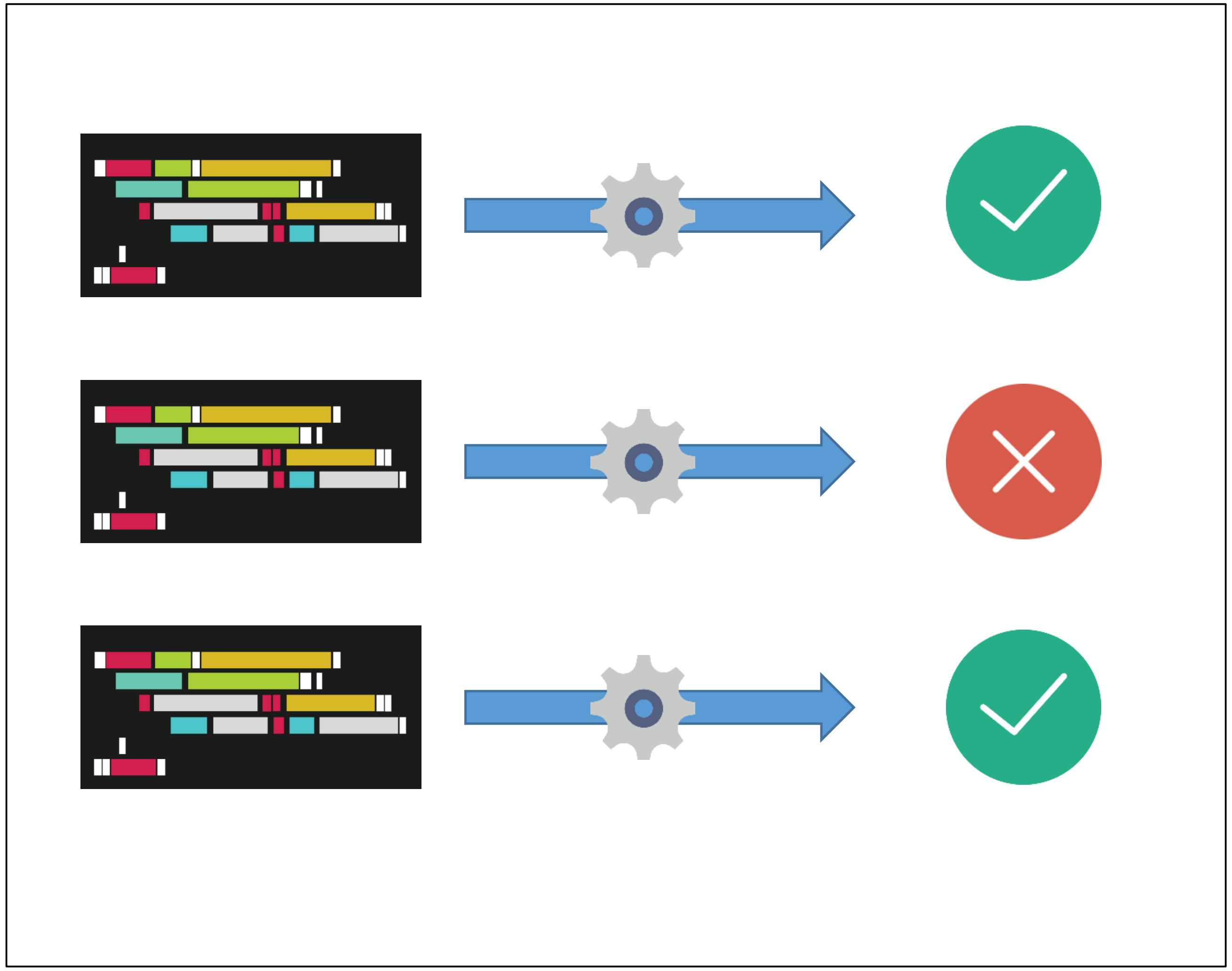
During this phase, we do the functional tests. A functional is basically done to determine whether the software does what it should do. Different types of functional tests can be identified. Examples are unit tests, integration tests, and system tests.



Lesson 6: Test

Unit tests usually already happen during the build process. A unit test is the test of an individual unit, or component, of the software. A unit is the smallest testable part of the code.

Let’s refer back to writing the code. We do this together, everyone writes small parts. How do we know if these small parts function properly? By having unit tests!

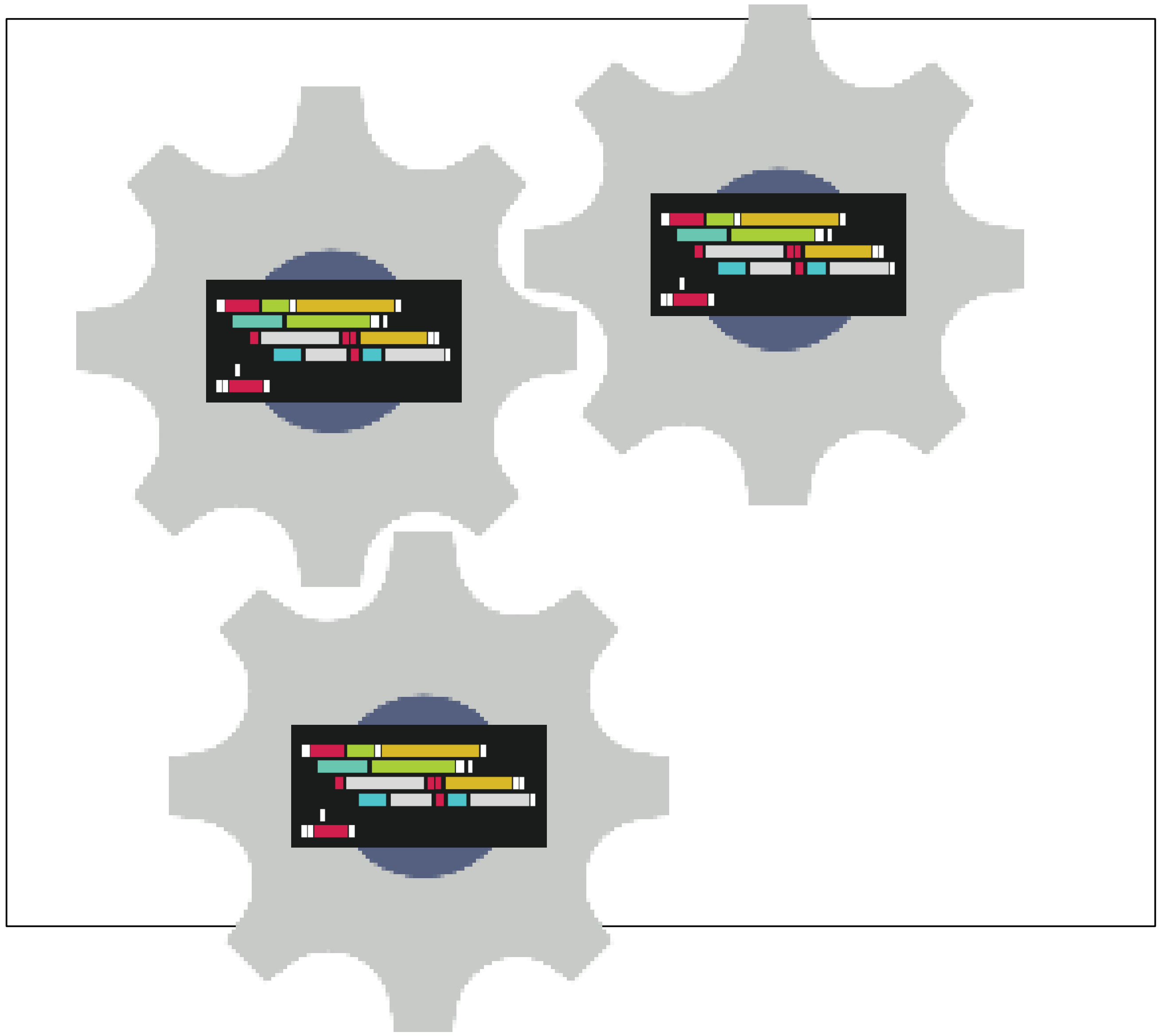


Lesson 6: Test

So, let’s say that all unit tests were successful. Yay! Now our code is fit for becoming one big application, right? Well, I might have to disappoint you.

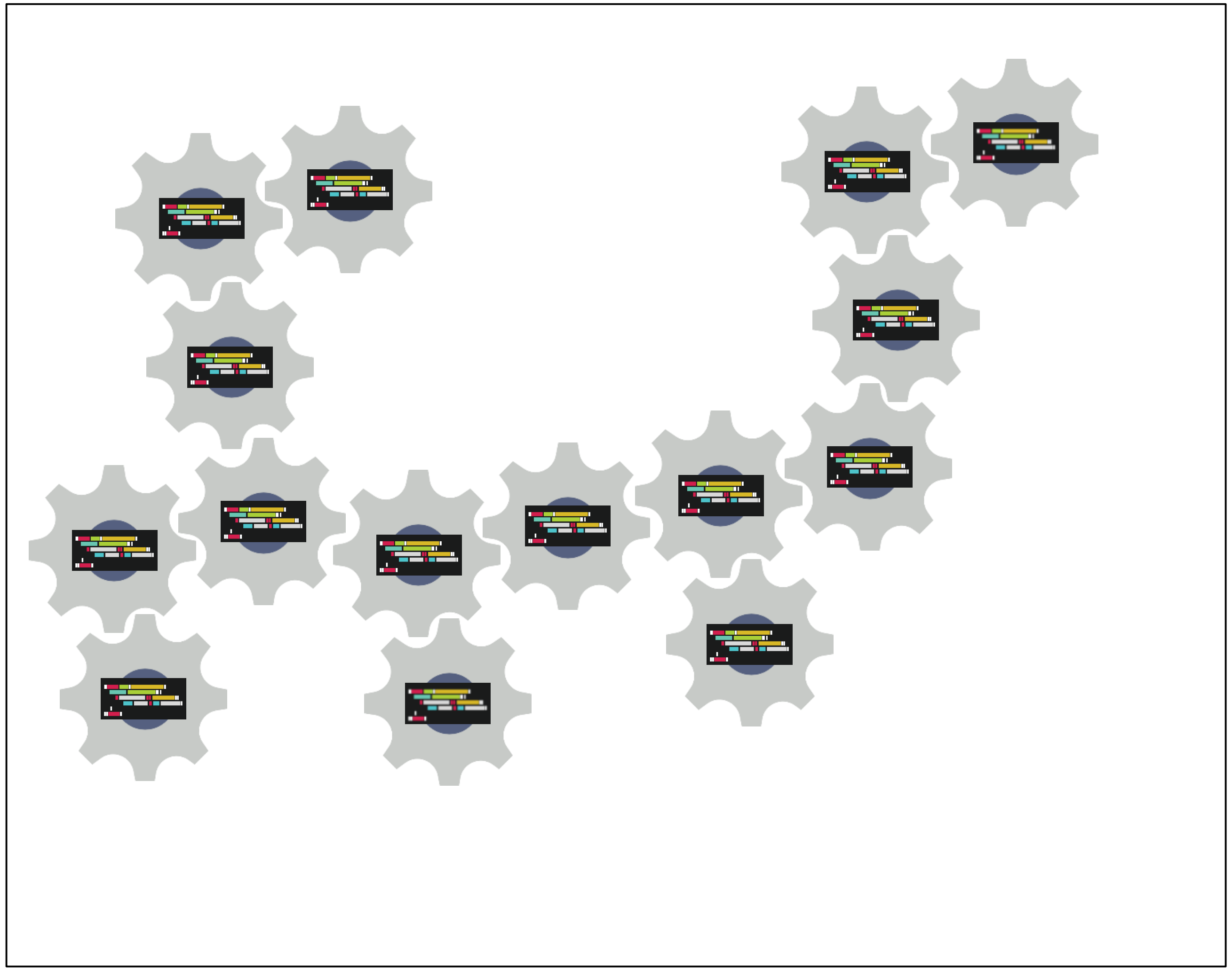
The fact that all units of code work fine on their own does not necessarily mean that they also function well together. And that is why we do integration tests.

With an integration test, the individual units are combined and tested as a group. If some units do not interact properly, we will find out now!



Lesson 6: Test

As you might already expect, we are not there yet. After the integration tests, we still need to do system tests. We combine all the groups that successfully made it through the integration tests and test the complete software.

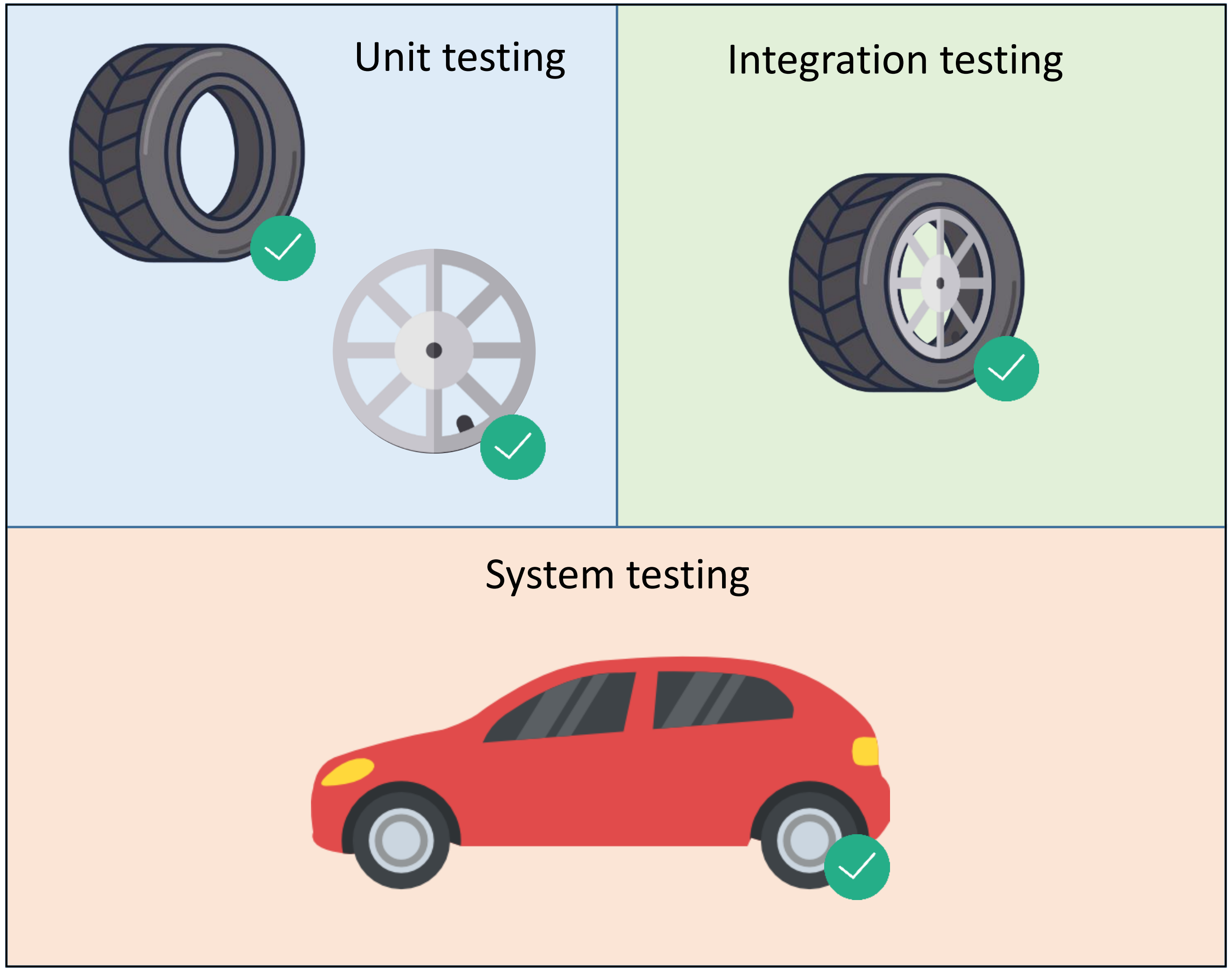


Lesson 6: Test

Let’s compare the functional testing process to the creation of a car. First, several units are made, such as nuts and bolts, rims, and tires. All these units are unit tested to see if they fulfill the function that they should.

Then, several units are integrated into a bigger part, for example the rims and tires to get to a wheel. This wheel is integration tested to see if the different units function together in such a way that we have a proper wheel.

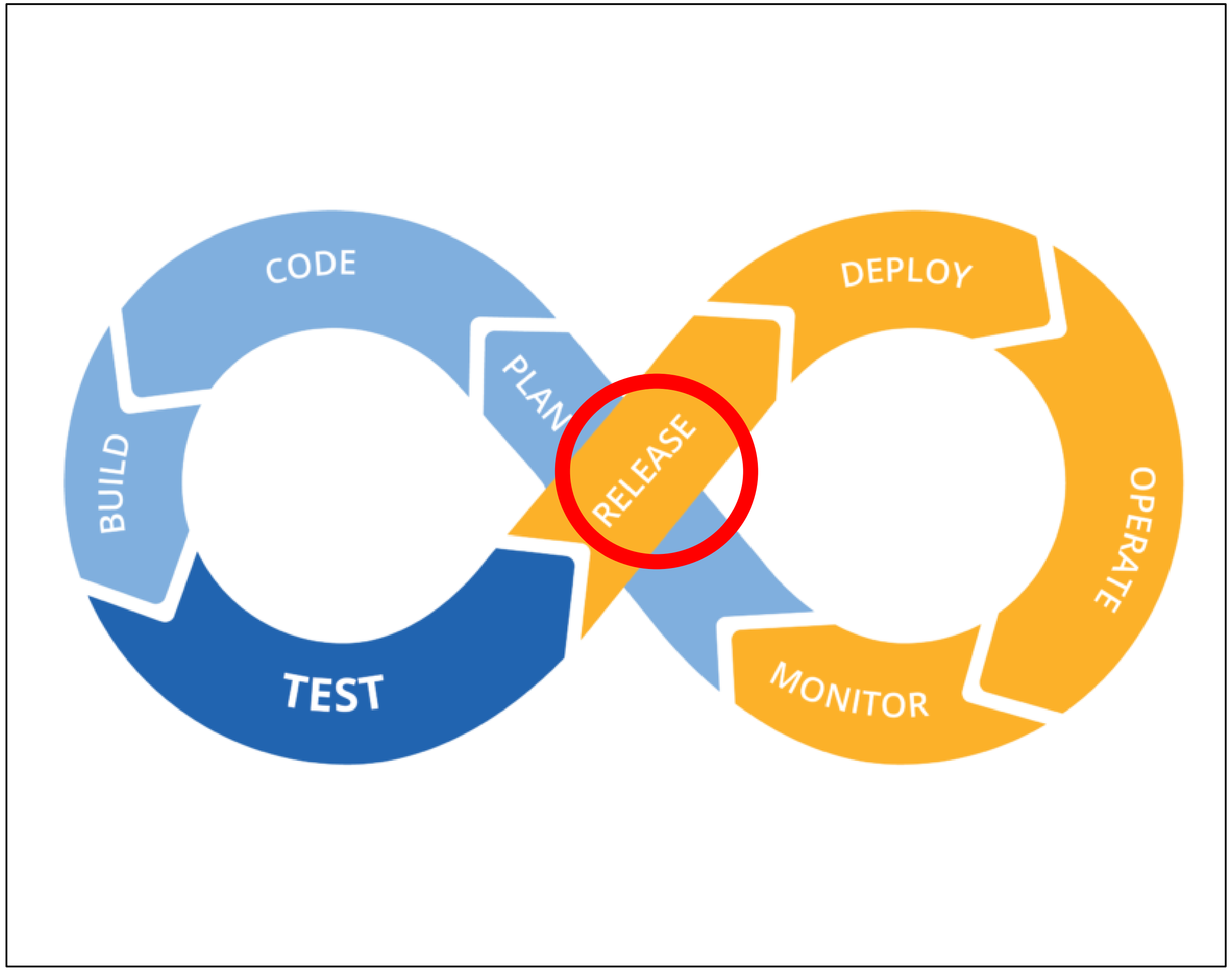
Finally, all the different integrated parts are combined into what we can call a car. The wheels, the engine, the seats, et cetera. This entire car undergoes a system test to see if all the different components work well together and if it can be driven safely.



Lesson 7: Release

Lesson 7: Release

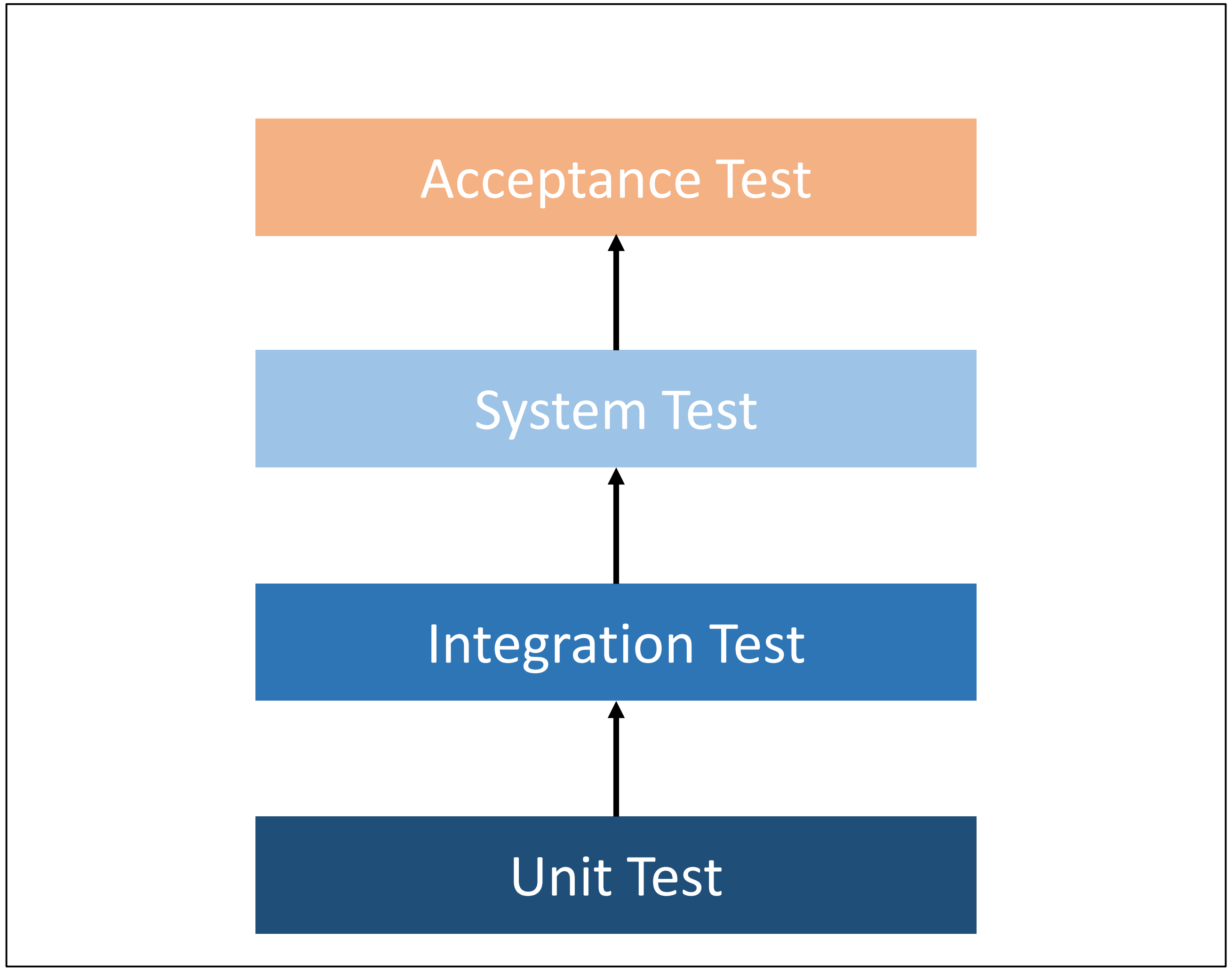
The next DevOps phase is the Release phase. During this phase we determine if the software meets the (non) functional requirements that were set at the beginning of the cycle, during the planning phase.



Lesson 7: Release

So, how do we determine if the software meets the (non) functional requirements?

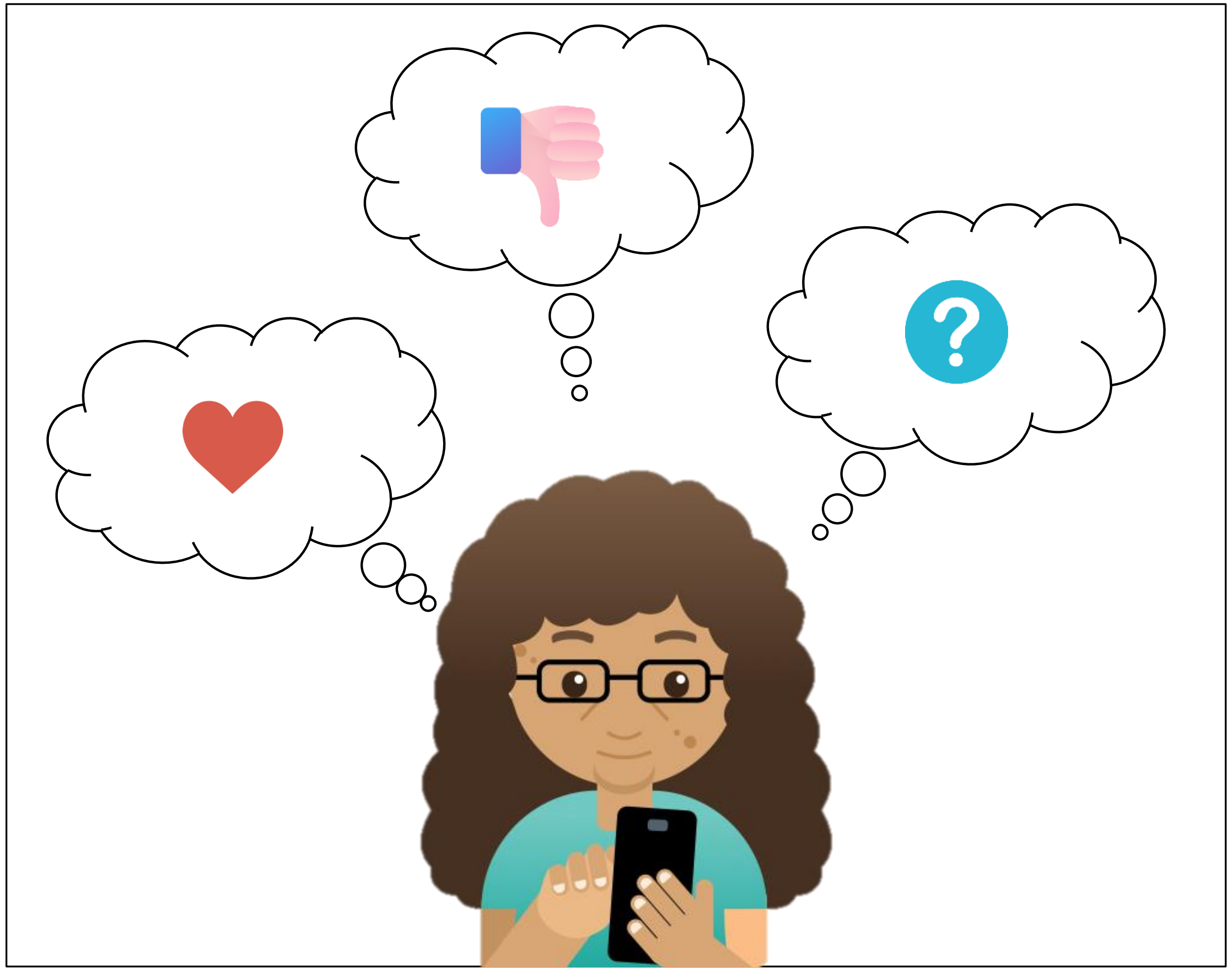
We basically already did this by having the functional tests. However, there might be one more test that is needed to be entirely sure: the acceptance test.



Lesson 7: Release

As the name already suggests, an acceptance test tests the software for acceptability during use. Therefore, it is sometimes also called a user acceptance test.

In fact, different types of acceptance tests exist: alpha & beta testing, contract acceptance testing, regulation acceptance testing, operational acceptance testing, and black box testing.



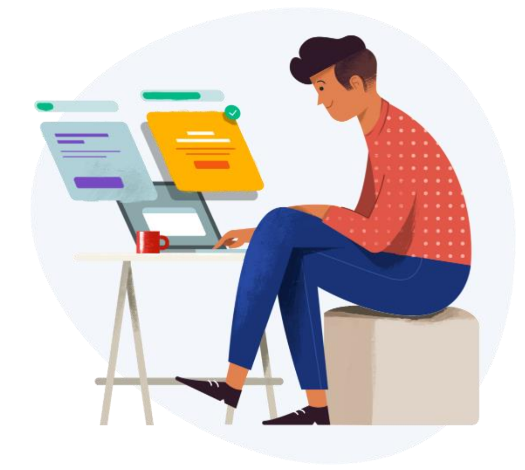
Lesson 7: Release

Alpha & beta testing, or AB testing, consists of basically two phases.

During alpha testing, the software is tested in the development environment. Different developers and testers might perform the test. The feedback will be used by the development team to improve the software.

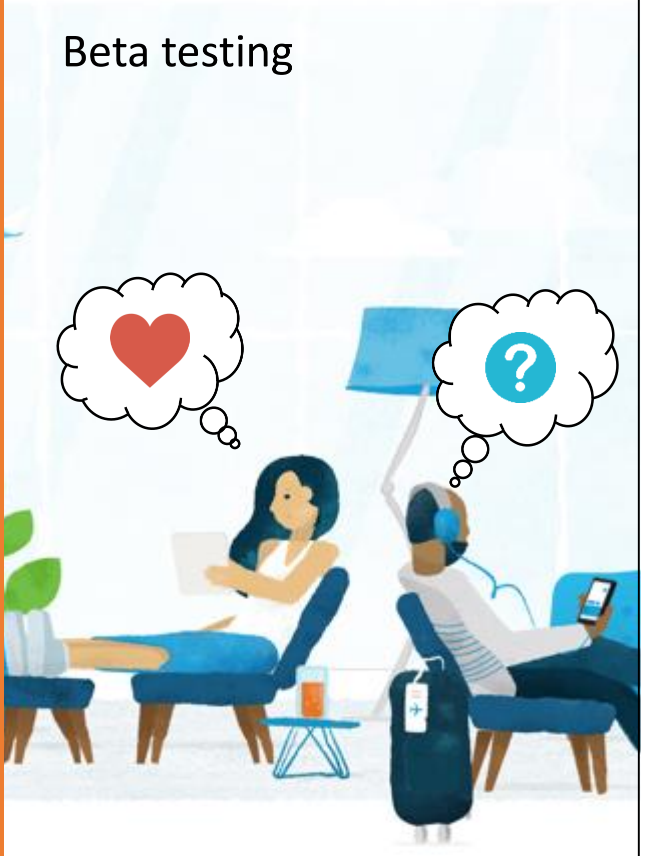
During beta testing, the software is taken out of the development environment and into the 'real world'. Potential end users are asked to interact with the software and to give their feedback. An example of this is going to the lounge at Schiphol Airport to test KLM's mobile app with real users who are present there. Of course, their feedback is taken into account for improvements.

Alpha testing



KLM Head Quarters

Beta testing



Schiphol Airport

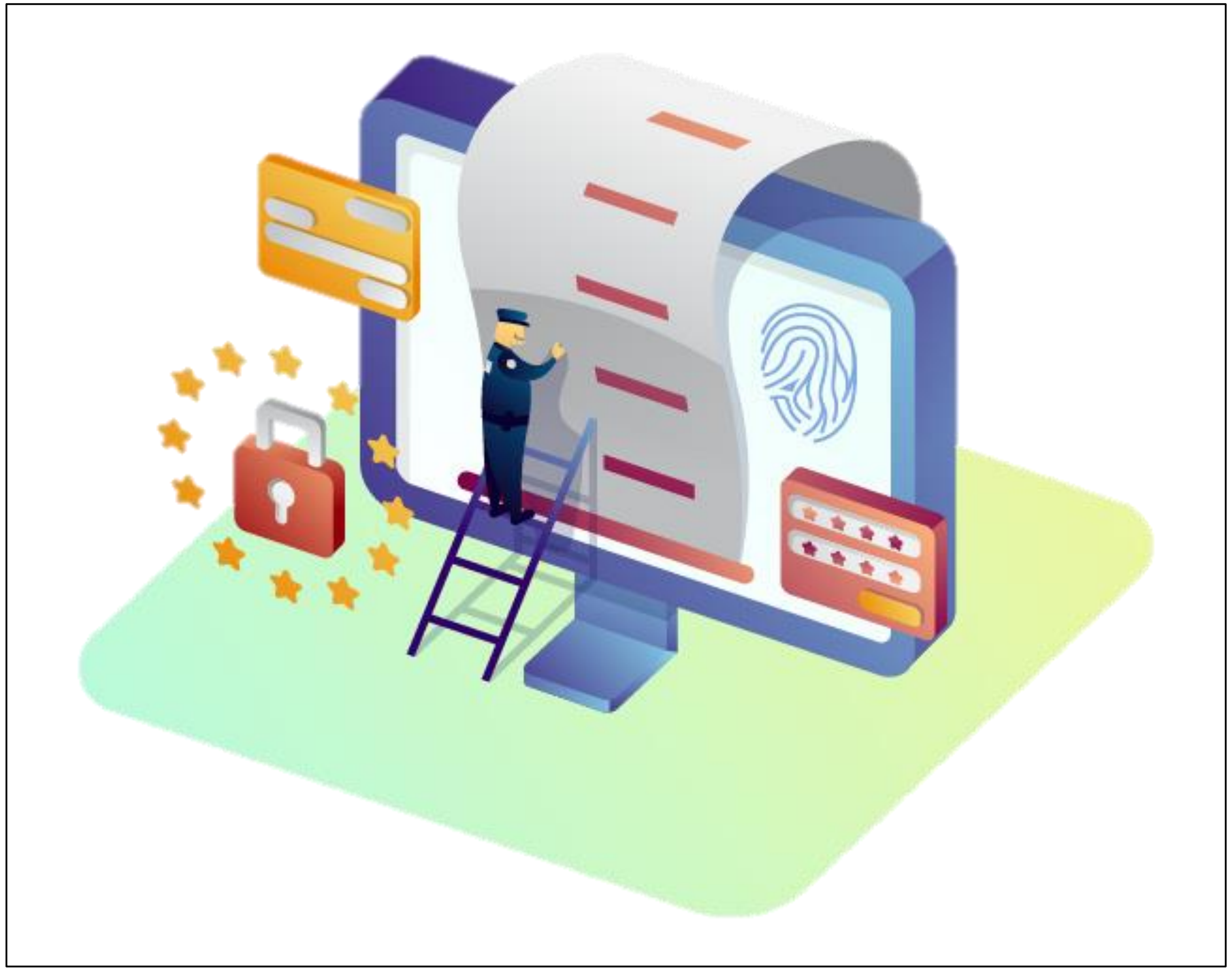
Lesson 7: Release

Contract acceptance testing is important when the development team has signed a contract with a client. This test will be held to find out if the software complies with the contract.

At AFKL, this never happens, since we do not make software for external clients. Hence, no contract compliances need to be tested.

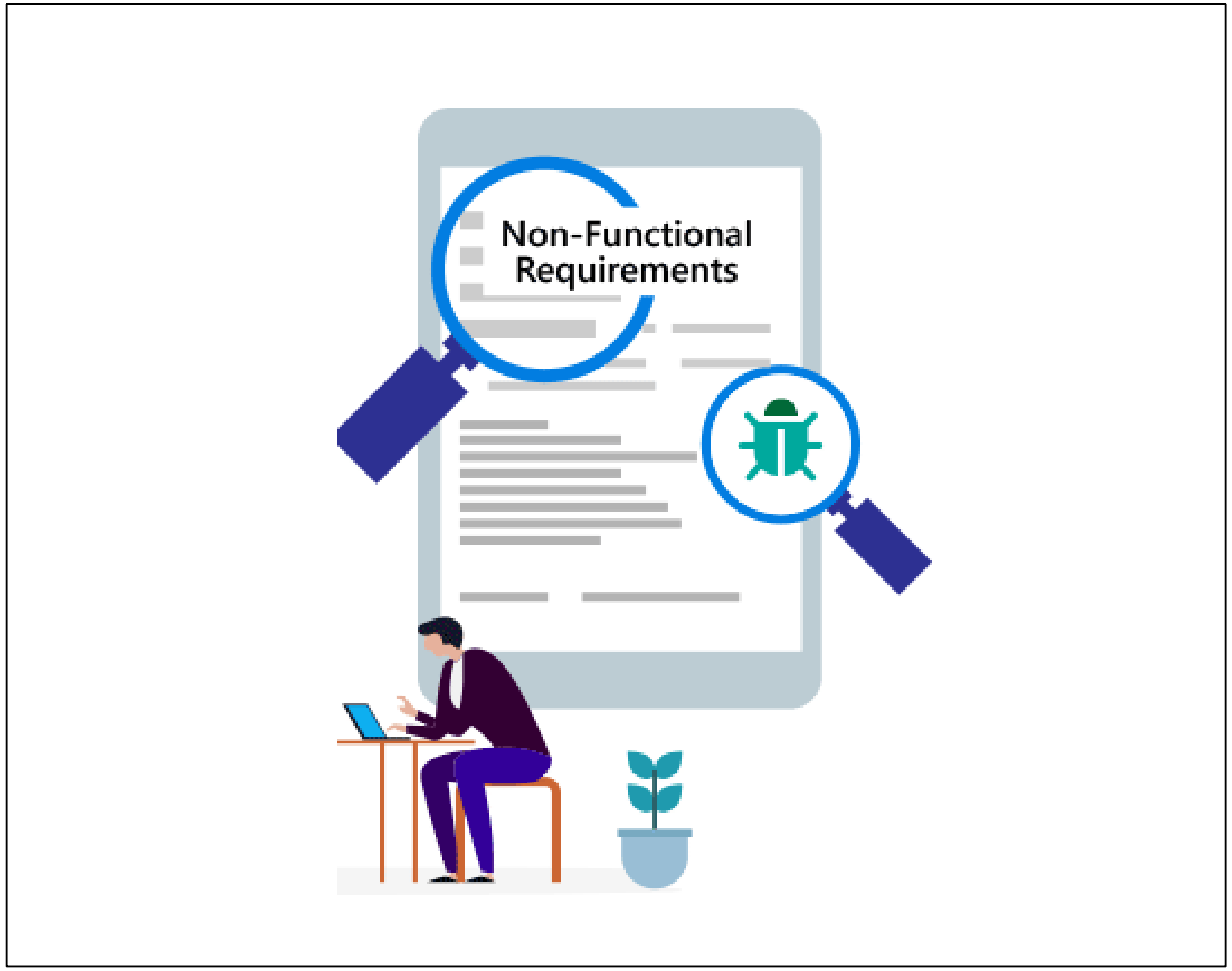
Lesson 7: Release

Regulation acceptance testing, also referred to as compliance acceptance testing, is done to test whether the software complies with legal regulations. Examples of legal regulation that should be adhered to are GDPR or PCI compliancy. Go to *Topic 16: Legal Regulations* to find out more about this.



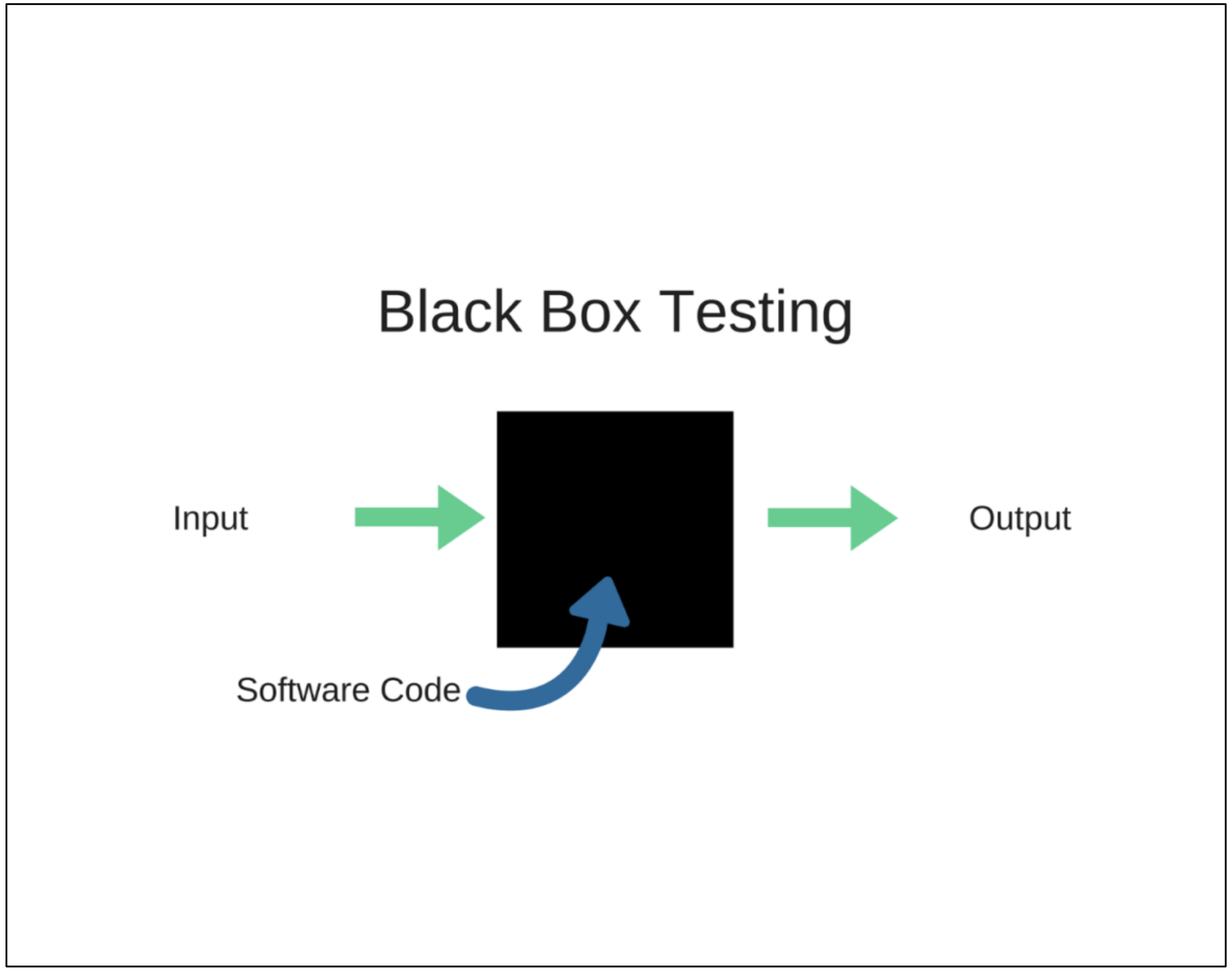
Lesson 7: Release

Operational acceptance testing, or operational readiness testing, is executed to ensure that the software can be used (operated) in an acceptable way. This is a non-functional requirements test which could, for example, focus on security or load on the application.



Lesson 7: Release

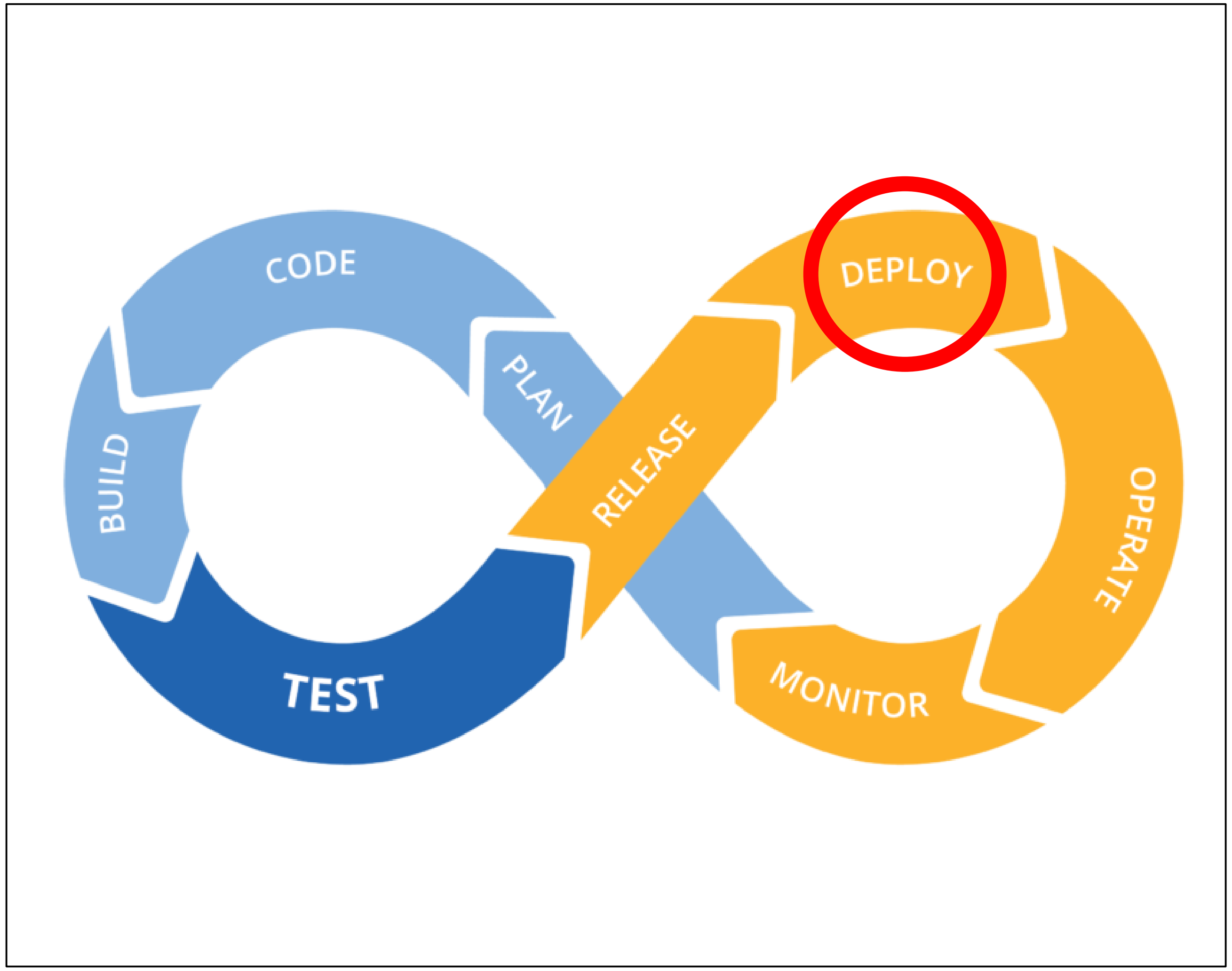
During black box testing the software is regarded as a black box. This means that the tester is not aware of the application code, it is like a black box to the tester. Instead, the tester is focussed on giving the software certain inputs and determining whether the outputs that it gives back meet the functional requirements.



Lesson 8: Deploy

Lesson 8: Deploy

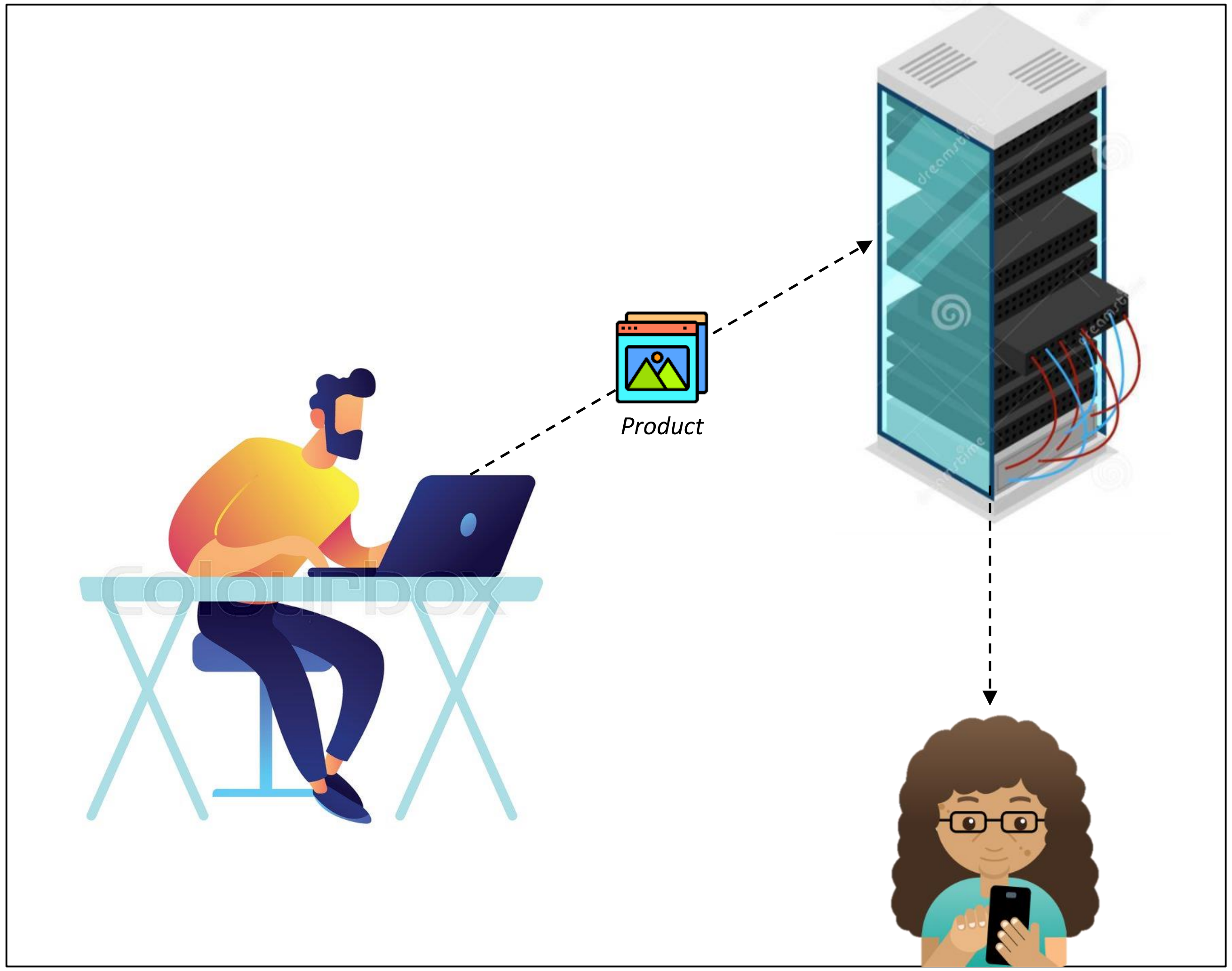
When all the previous phases have been completed successfully, the software can finally be given to the user! This is what happens during the Deploy phase.



Lesson 8: Deploy

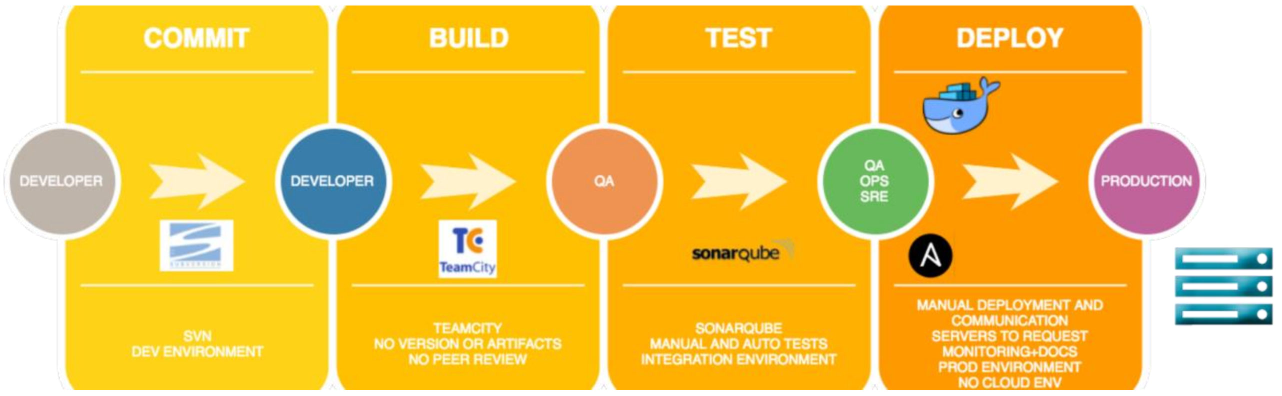
So, “giving the software to the user”, how do we do that? It is actually quite simple. We install the software that we coded, built, and tested on a server, and from then on the user can access it.

If we have, for example, been developing something new for the KLM website, the user can only see it on the live website once we deployed the code to the server. Therefore, we say that the software is “live” once we have deployed it.



Lesson 8: Deploy

We wouldn’t be still talking about DevOps if we didn’t try to automate the deployment process as much as possible. This is what we call the deployment pipeline, or the delivery pipeline at AFKL. This will be addressed further under *Topic 11: Delivery Pipeline.*

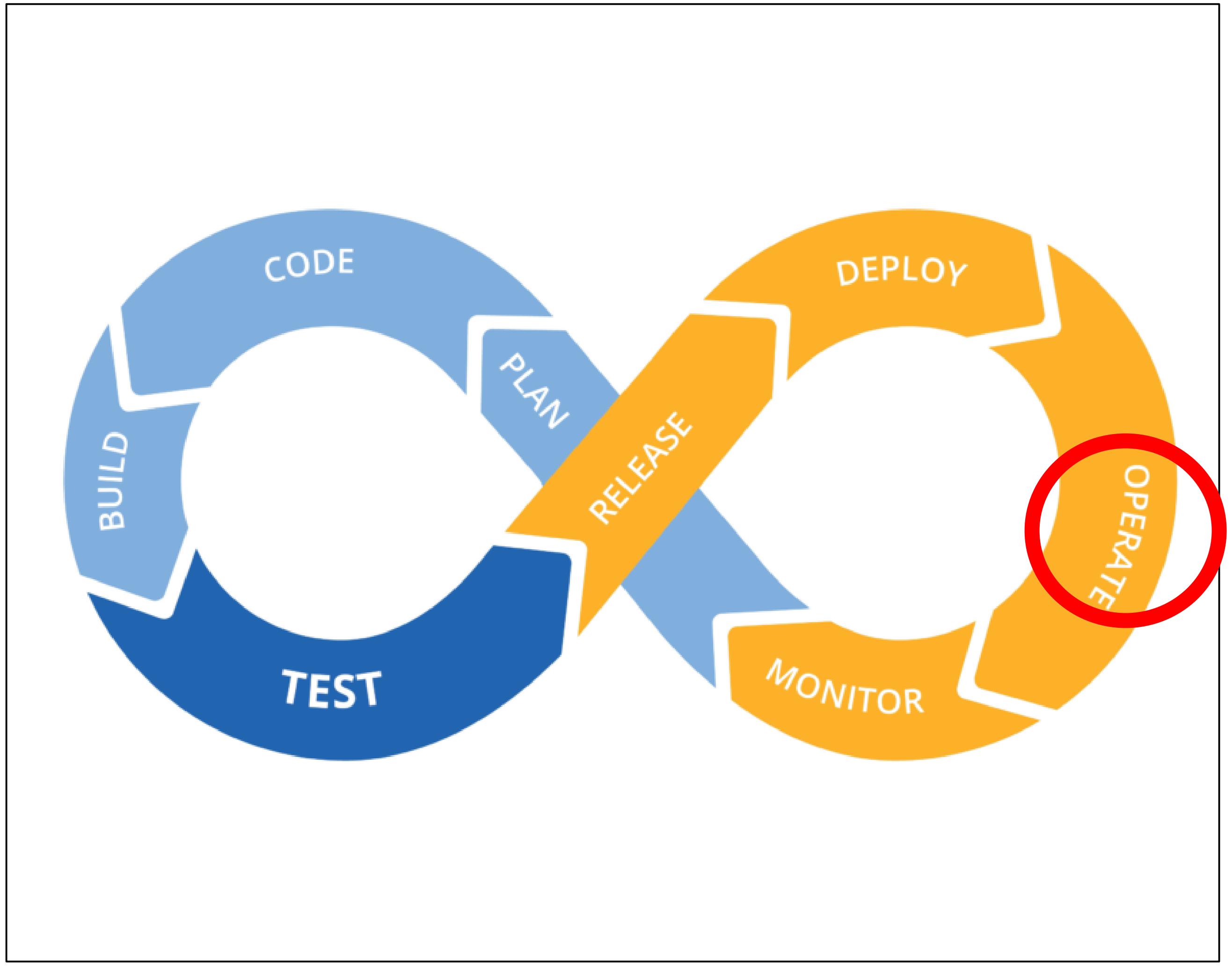


Lesson 9: Operate

Lesson 9: Operate

Alright, the software has been created and given to the customer, guess we are done!

Well, not exactly. Don't underestimate the effort it takes to keep the software up and running. This is what happens during the Operate phase.



Lesson 9: Operate

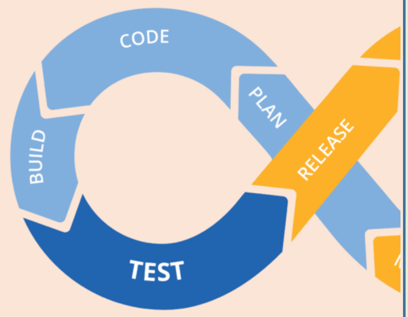
Keeping your software up and running is quite important if you want to keep your customers happy, but also if you want to make revenues.

Imagine you, as a customer, want to buy a plane ticket for your vacation but AFKL's website is down. That would be a shame! You have to find another airline's website to book your tickets and AFKL would not make any revenue. Sounds like a loose-loose situation. Unfortunately, these things happen if the operate phase is not taken into account properly.

Lesson 9: Operate

So how do we do this? Well, actually there are multiple processes involved: infrastructure management, application management, information management, and incident management.

Infrastructure management



Application management

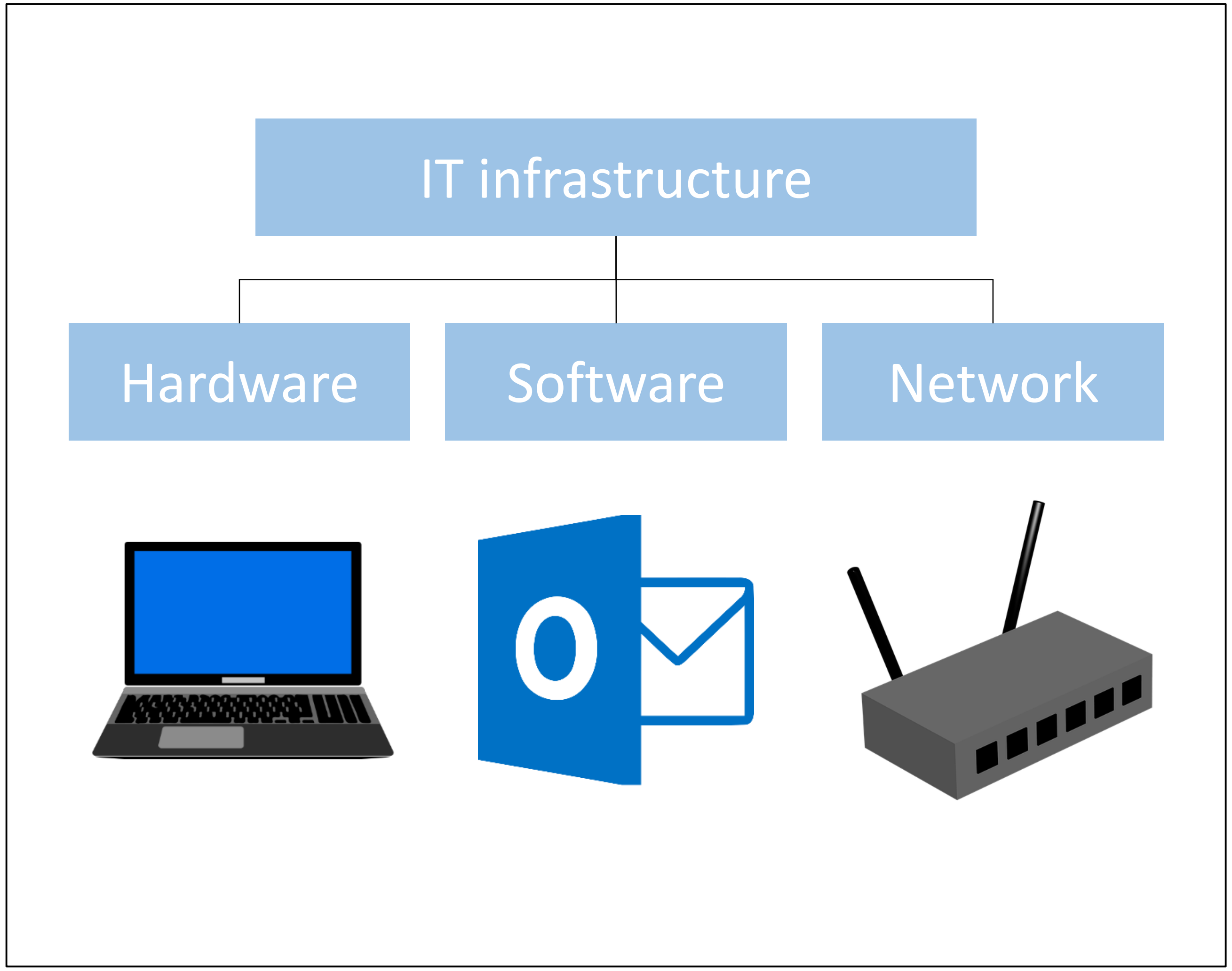
Information management

Incident management

Lesson 9: Operate

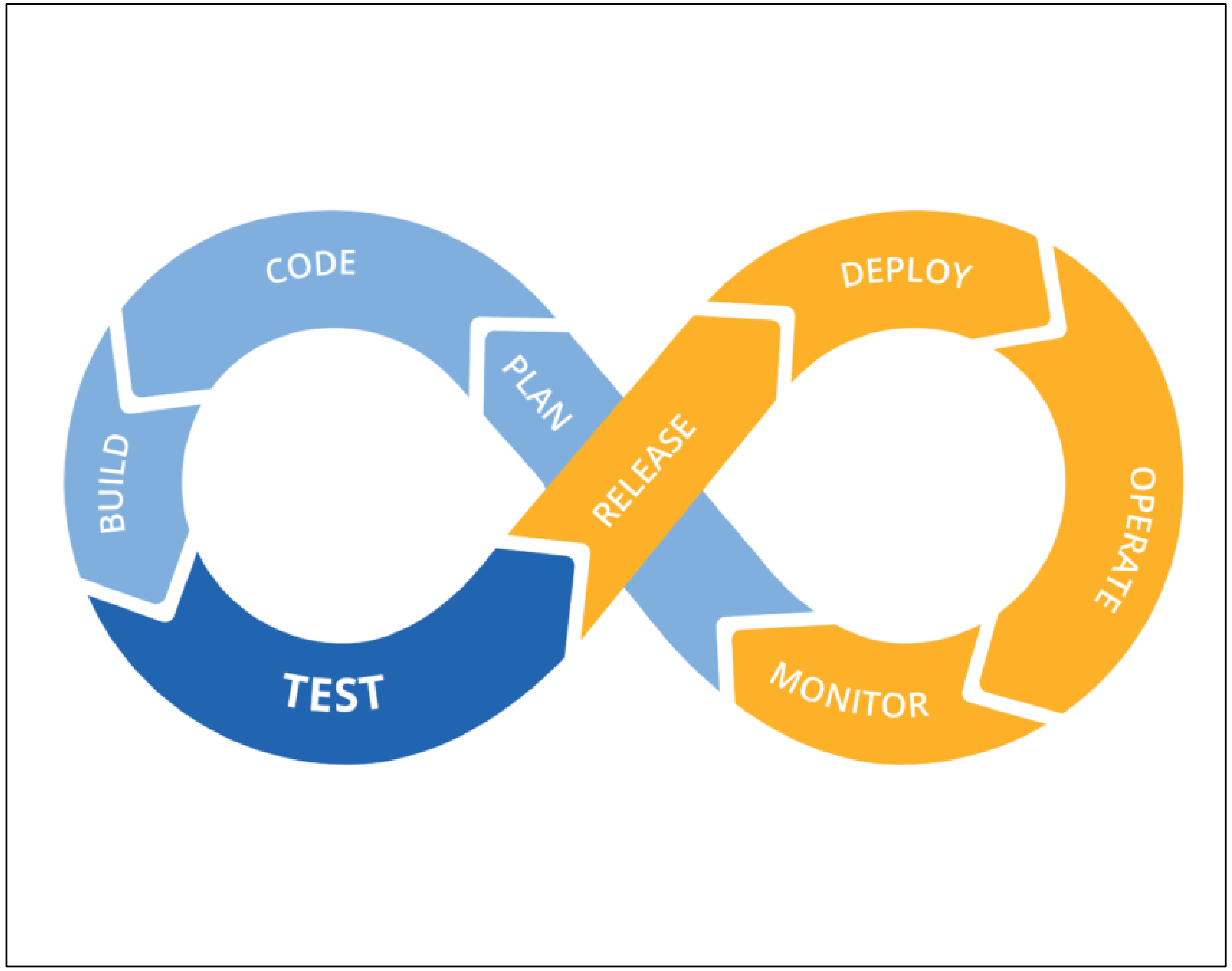
As the name already suggests, infrastructure management deals with the IT infrastructure. Unfortunately, IT infrastructure is a rather broad term and quite difficult to grasp. It can mean anything from hardware, software, network components, and even much more in the IT environment that should be in place in order to be able to provide IT services, such as your application.

This might sound like a distant thing to you, but actually you are using IT infrastructure all the time. Think about sending an e-mail. Part of the infrastructure are your laptop or phone (hardware component), the e-mailing application (software component), and the router (network component).



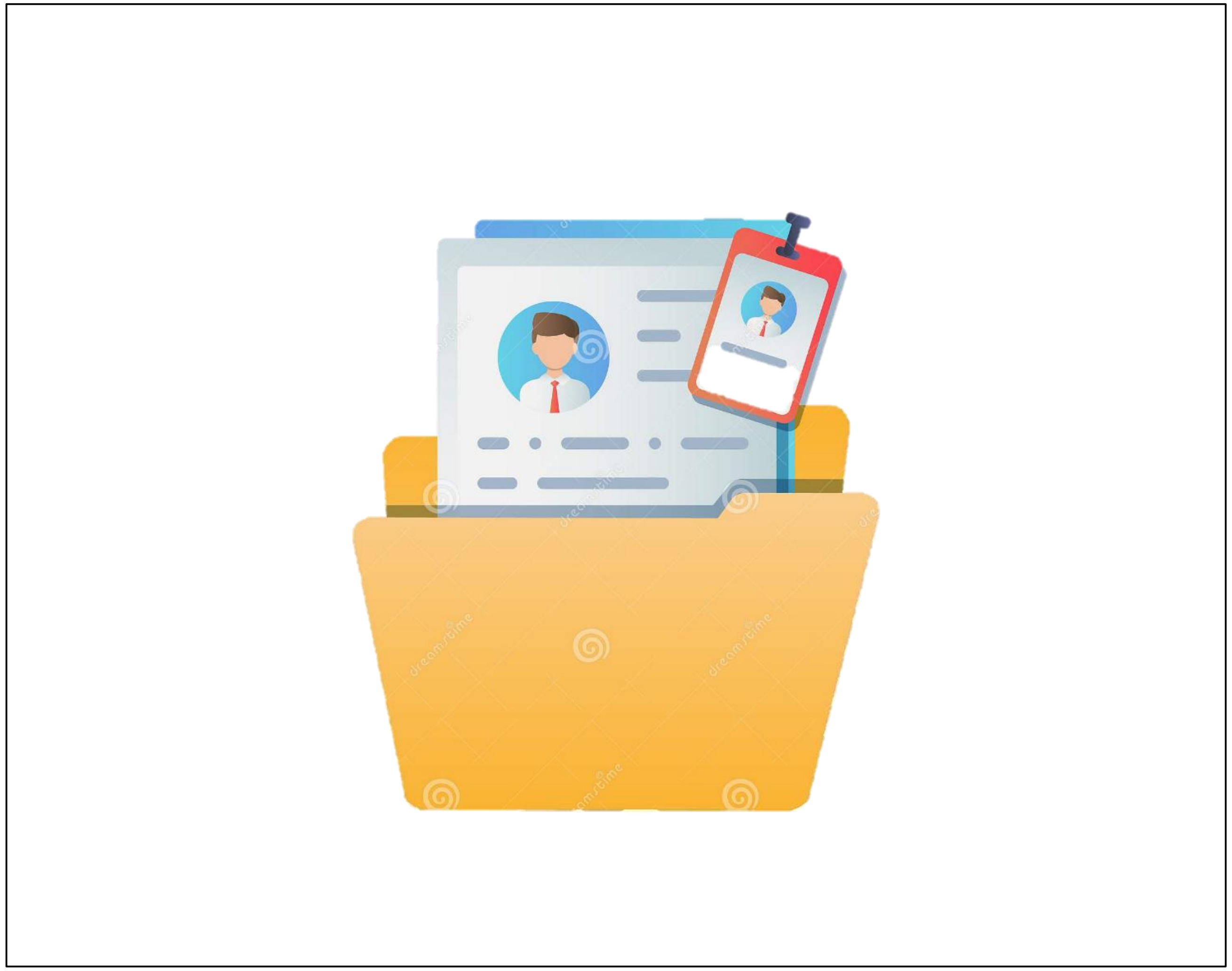
Lesson 9: Operate

Application management is about managing the application throughout its lifecycle. As you might realize by now, the application lifecycle is basically what we are discussing through the DevOps phases. So, actually we were already doing this.



Lesson 9: Operate

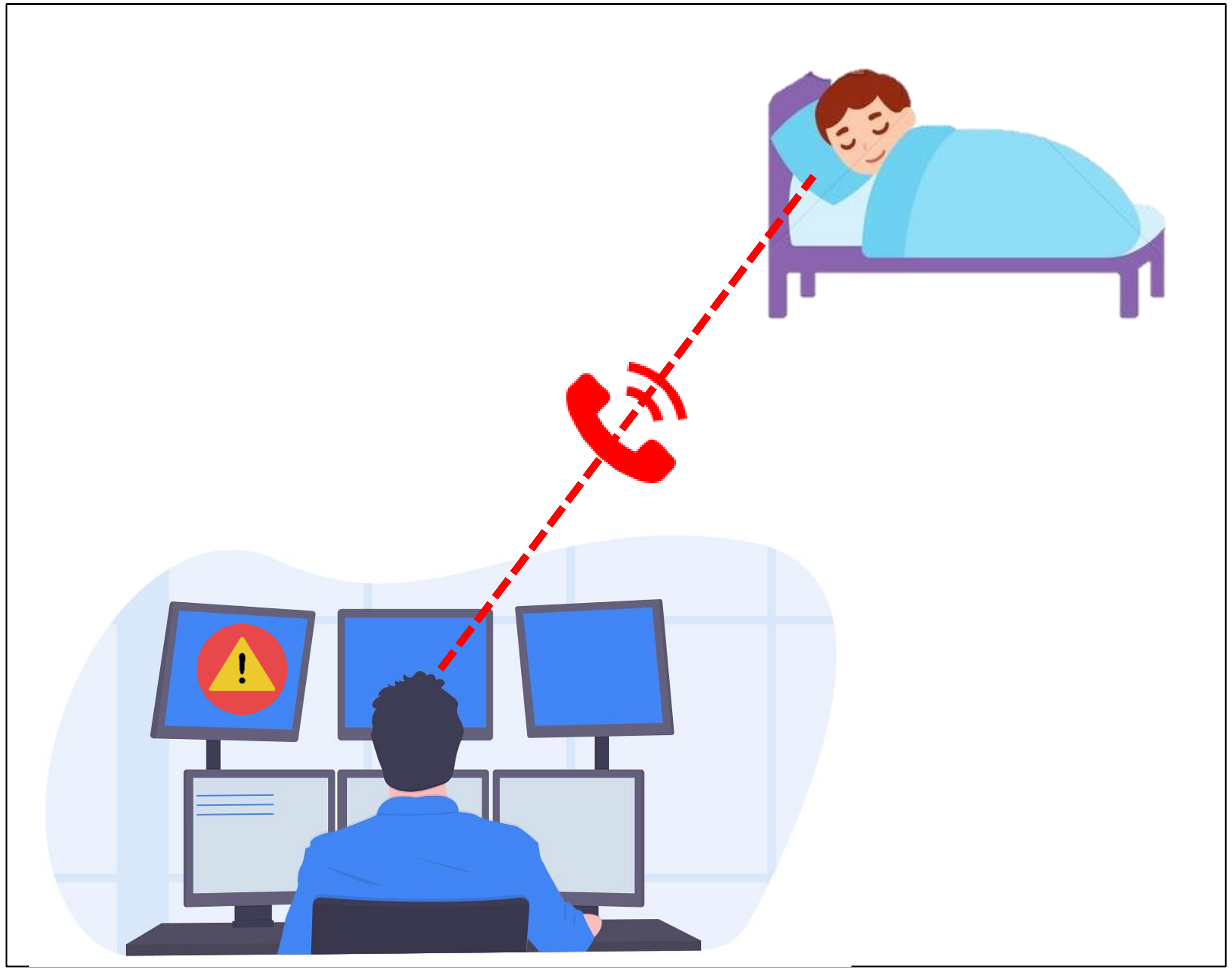
Information management is about managing the data in the application. For AFKL specifically, this might be about data concerning flight times, weather forecasts, pilots, employees, or passengers. As you might expect, this is a lot of data and some of it is very confidential. How to manage the huge amounts of data and how to keep it safe are things that information management is concerned with.



Lesson 9: Operate

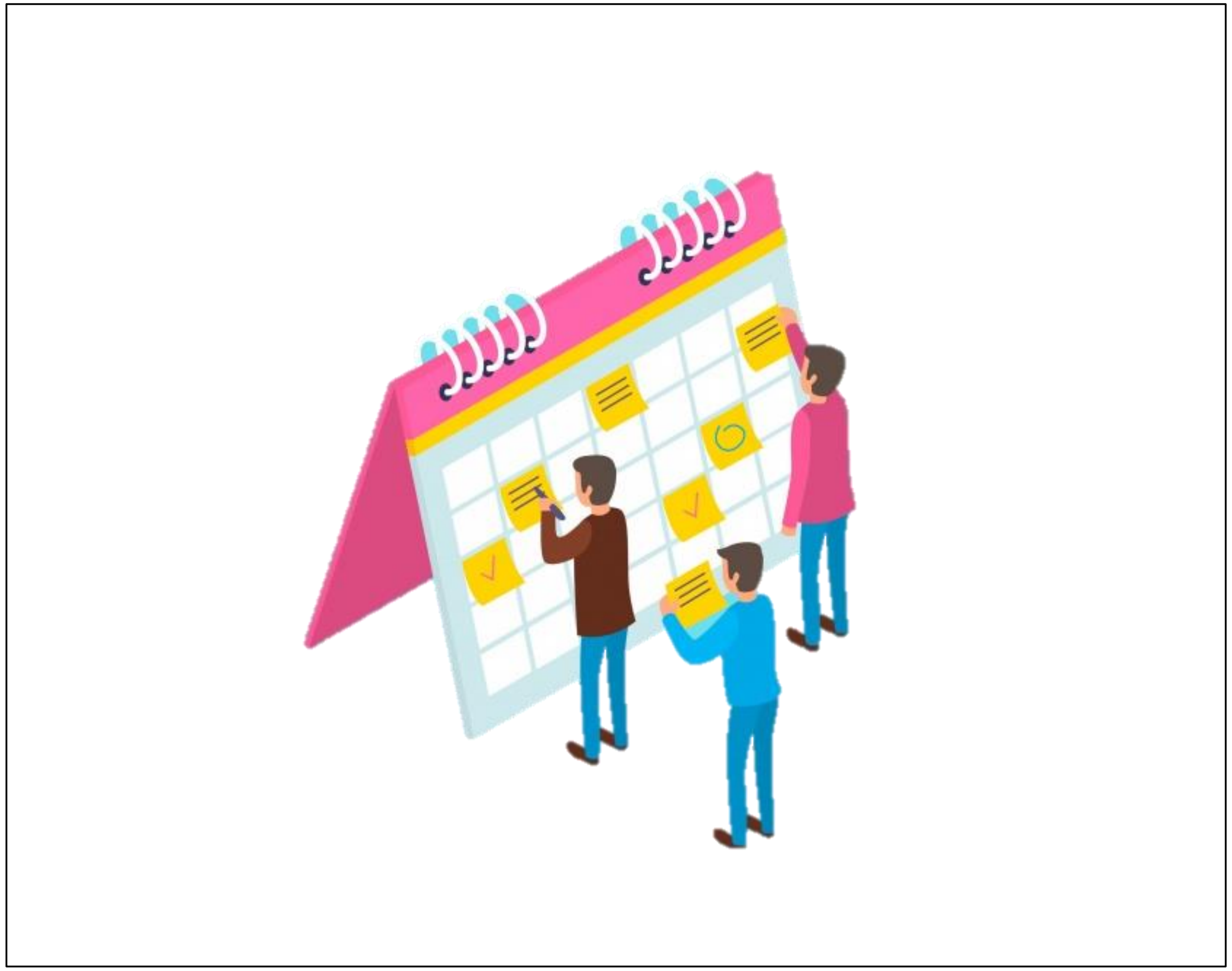
At AFKL, the most important process during this phase is Incident Management. An incident is an event that disrupts the application, and therefore disrupts the organization’s operation and services.

If an incident happens, it means that the team responsible for the application needs to solve this. Immediately. It is therefore possible that a developer will get a call in the middle of the night saying that an incident is found and needs to be solved. You can go to *Topic 19: Incident Management at AFKL* to find out more about this.



Lesson 9: Operate

As you might begin to realize, these three processes should not be started once our application is live. Instead, they should be taken into account from the beginning of the application life cycle already.

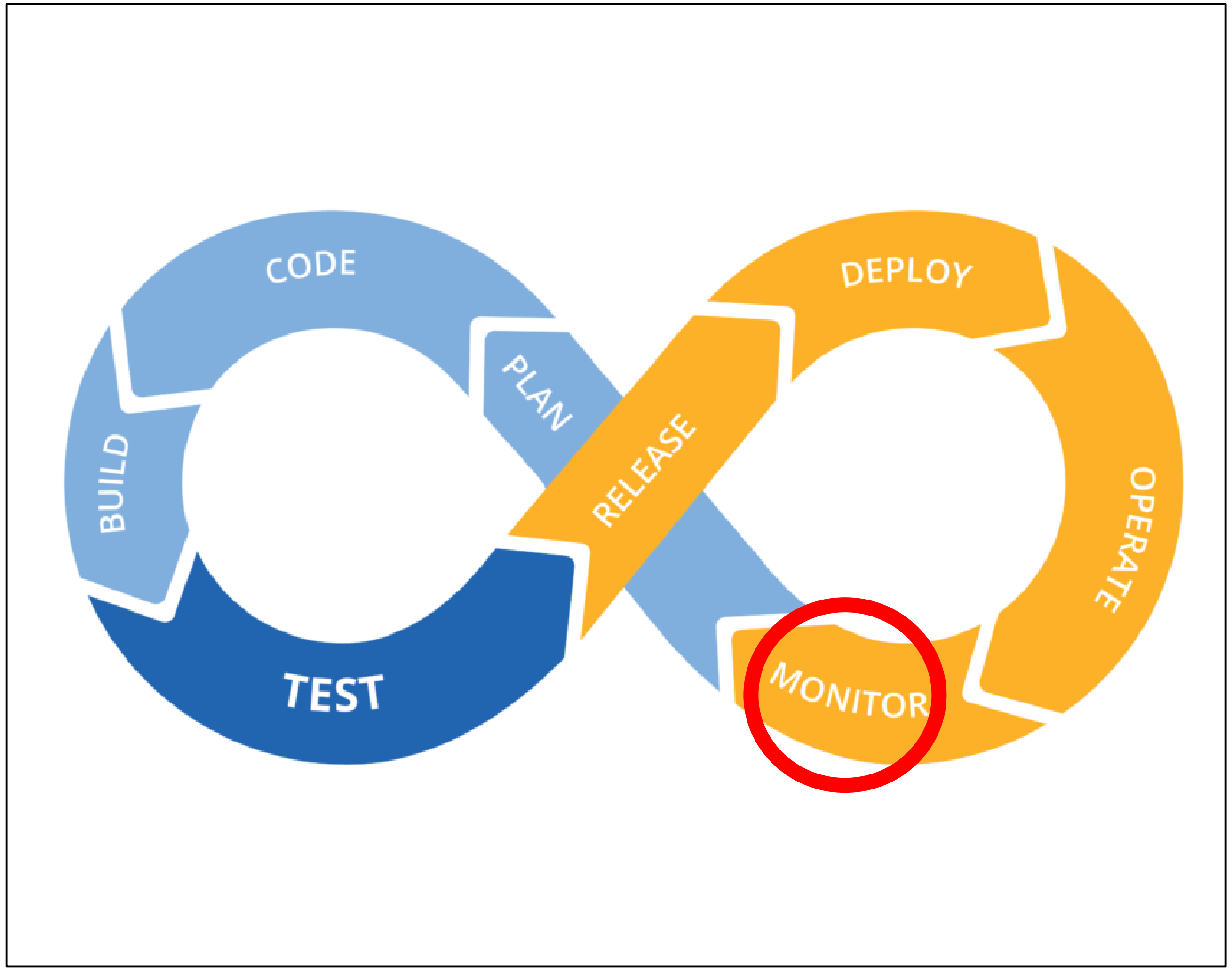


Lesson 10: Monitor

Lesson 10: Monitor

The final phase to discuss is the Monitor phase. We already saw that the Operate phase is concerned with keeping the software application up and running. That's really nice. But how do we know when there is something wrong with the application, something that prevents it from being up and running (or might potentially prevent it at a later time)?

Well, that is why we monitor. We keep an eye on the software application to see if any incidents occur. If not, perfect. Otherwise, we obviously have to do something about this.

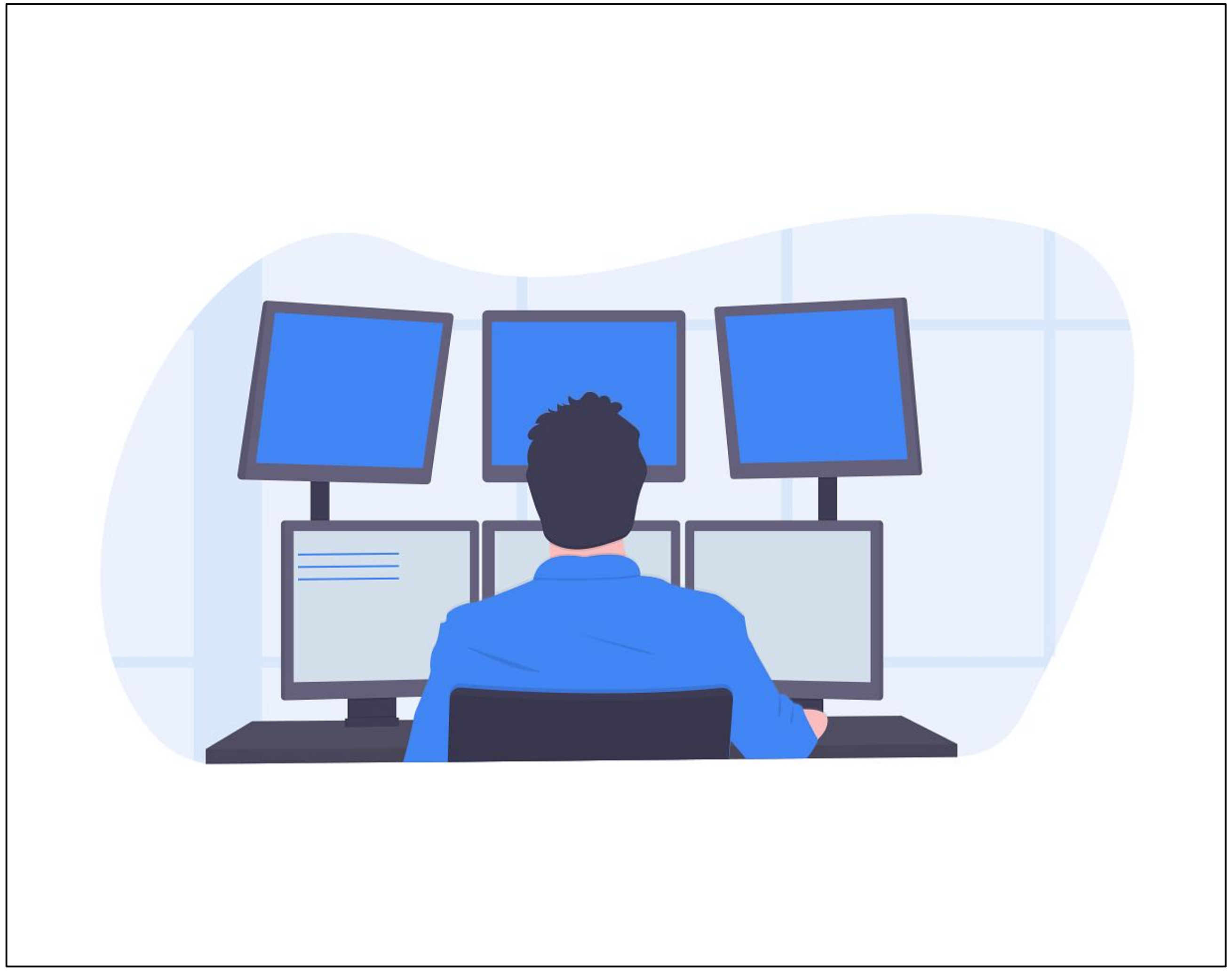


Lesson 10: Monitor

There are basically two kinds of monitoring: technical monitoring and business monitoring.

Lesson 10: Monitor

Technical monitoring is about monitoring the application from a technical perspective. During technnical monitoring we focus on events and incidents. An event is something that can occur in the application, while an incident is an event that negatively affects the application. If you want to find out more about this, please go to *Topic 19: Incident Management at AFKL.*



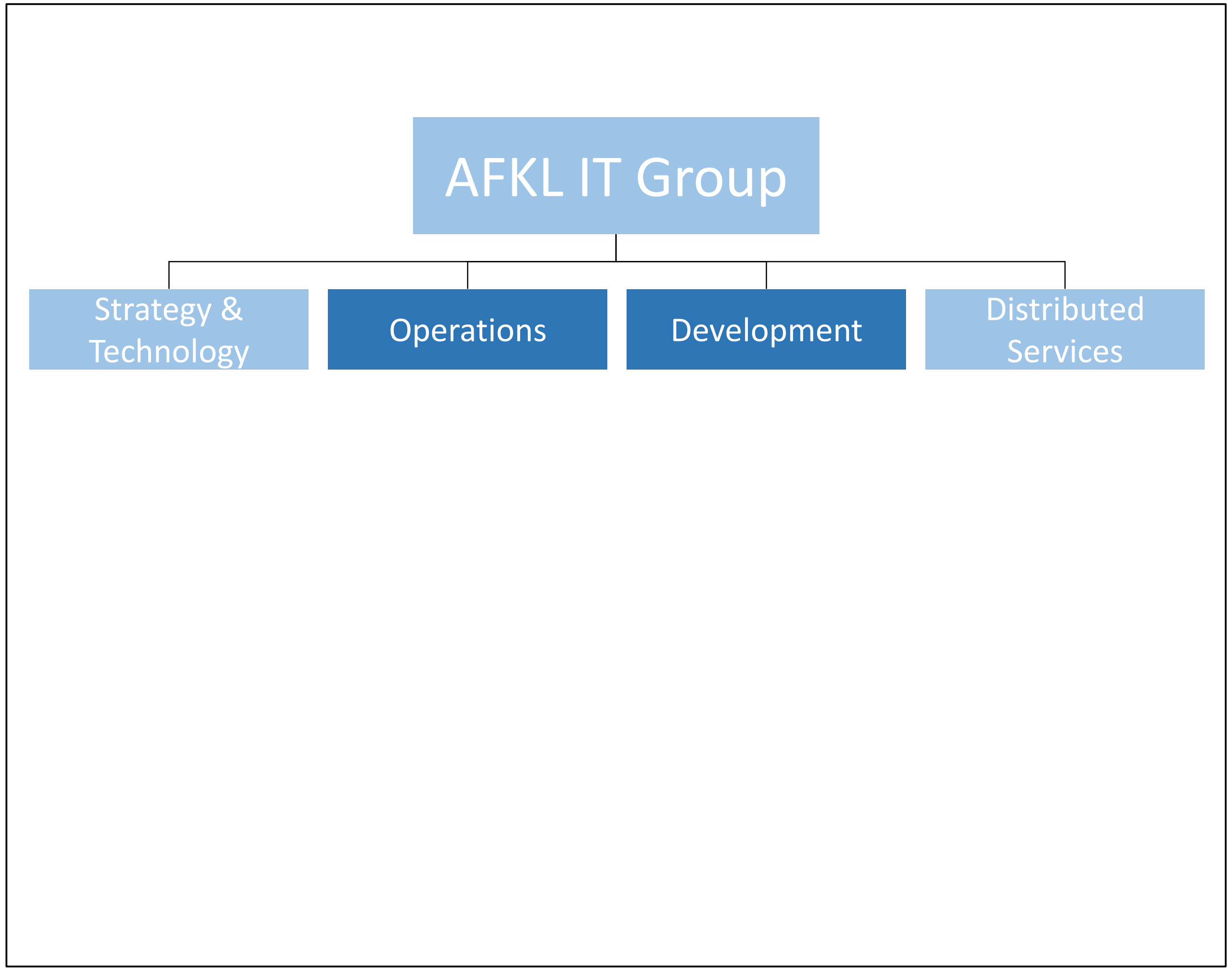
Lesson 10: Monitor

Business monitoring is a bit different from technical monitoring, as it looks at the business side of the application. Examples are monitoring the amount of visitors on the website, or the behaviour of customers who book flights. From this data, we might be able to make better decisions about what we should offer through our applications and how.

Lesson 11: DevOps at AFKL

Lesson 11: DevOps at AFKL

So, you might be wondering, how is DevOps practiced at AFKL? Well, AFKL still has Development and Operations as two different departments (if you want to know more about this, have a look at *Topic 6: IT Departments*). Sounds like we are still working in the old silos…



Lesson 11: DevOps at AFKL

Luckily, this is not true. AFKL is actively stimulating its development teams to adopt the DevOps way of working. How? By stimulating an Agile way of working in the product teams, by offering the teams Self Services, and by offering tooling for an automated delivery pipeline.

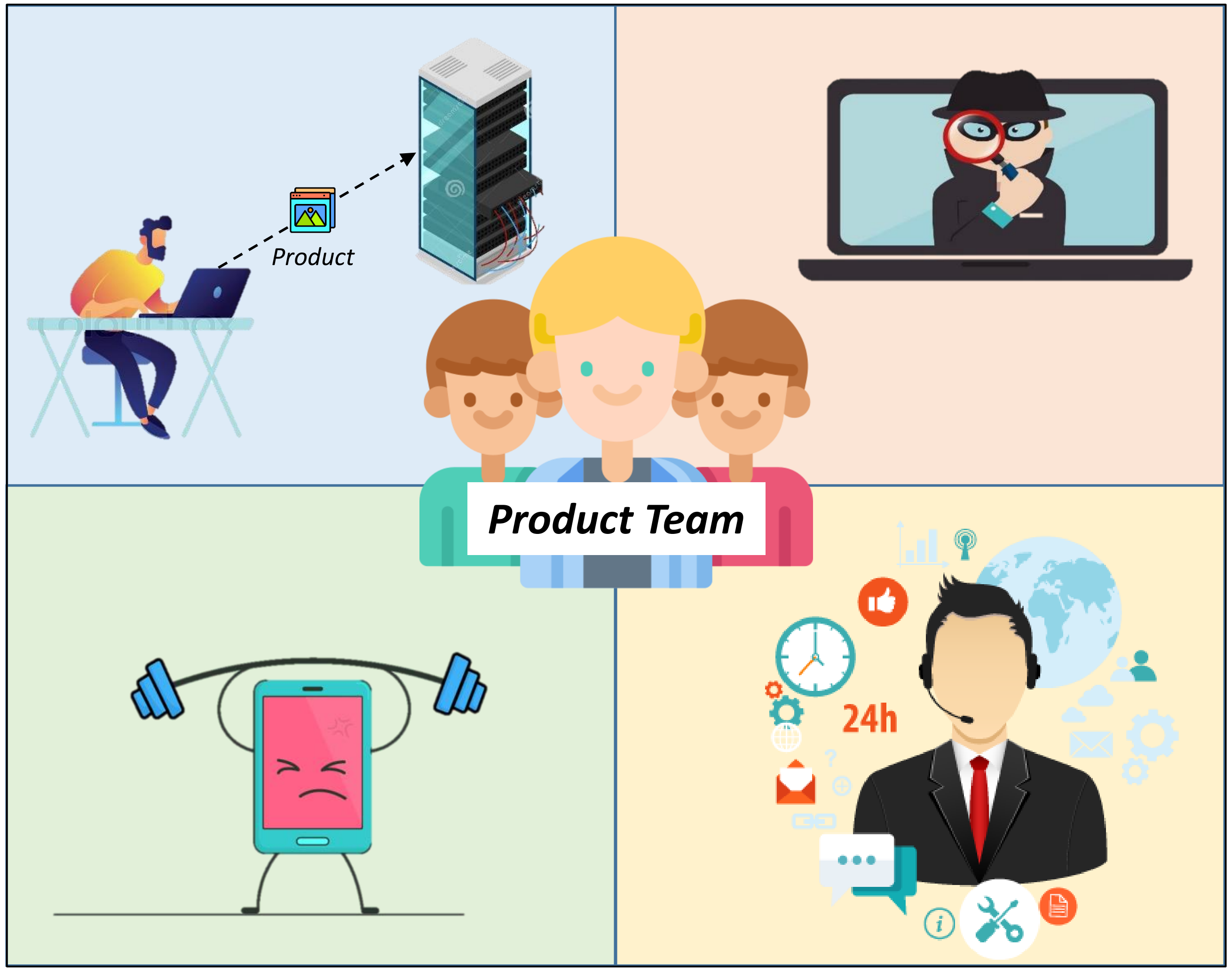
Lesson 11: DevOps at AFKL

There are multiple ways in which AFKL tries to stimulate the Agile way of working. For now it might be most important to know that the development of our B2C software applications is done in small product teams which work in short cycles according to Scrum and SAFe. If you want to find out more about this, have a look at *Topic 1: Scrum* and *Topic 2: SAFe*.

Lesson 11: DevOps at AFKL

AFKL's Self Services can be described as "the pragmatic approach to DevOps within AFKL". Self Services are tracks that stimulate AFKL's product teams to take responsibility of more and more tasks that were traditionally done by Operations. These tasks include setting the application to live, security scans, performance tests, and application support. You can find more information about this under *Topic 10: Self Services*.

The Operations department is still accountable for these tasks, but the aim is that the product teams learn to do more and more of it by themselves.



Lesson 11: DevOps at AFKL

And last but not least, the tooling for an automated delivery pipeline. As said before, DevOps is all about automating the processes around software development, delivery and maintenance. Part of this is an automated delivery pipeline, which means as much as having an automated way in which the software is released.

Although this might sound easy, an entire pipeline has to be in place in order to make sure that things go smoothly. AFKL's DevOps department works on the tooling for the delivery pipeline that the teams have to work with. To learn more about this, please go to *Topic 11: Delivery Pipeline*.

Quiz