SOFTWARE ENGINEERING FOR DATA INTENSIVE SCIENCES

LIBFOARPDLMDSSEDIS01

LIBF

LEARNING OBJECTIVES

The course **“Software Engineering for Data Intensive Sciences”** gives a comprehensive overview of software engineering basics for practitioners coming from data intensive sciences, such as data science. In the first unit you will be introduced to two fundamental approaches to project management, namely traditional and agile methodologies. Both approaches can be found in companies today and come with their individual advantages and drawbacks. Whatever approach you find yourself using at work, project management is the basis of working productively in teams, and having a good understanding of best practices goes a long way. The second unit is concerned with a modern approach to combining software development and operations within a company, called DevOps. The DevOps culture is fundamental to how big players like Google deploy and maintain their services, and very popular among startups today. You will be introduced to the core building blocks of scalable software environments based on the DevOps philosophy. In unit three you will learn the basics of software development best practices, starting with how to test the software you write and how tests get automatically checked in a production pipeline. You will also learn about version control for your software and the tool stack professional developers use in practice. Unit four is about application programming interfaces (API), that is, the various ways in which you can interact with software and services. You will learn some of the most important principles of designing and building good interfaces on your own and build a small Python library based on them. The final unit introduces you to the differences between the workflow of data scientists and software engineers, how their requirements and expectations differ, and how the work of data scientists safely finds its way into scalable production environments.

UNIT 1

AGILE PROJECT MANAGEMENT

STUDY GOALS

On completion of this unit, you will have learned …

- the definition and types of traditional project management.
- what Agile project management is.
- what Kanban and Scrum methodologies are.
- which modern methodologies are used for project management.
- how to transition from traditional to Agile project management.

1. AGILE PROJECT MANAGEMENT

Introduction

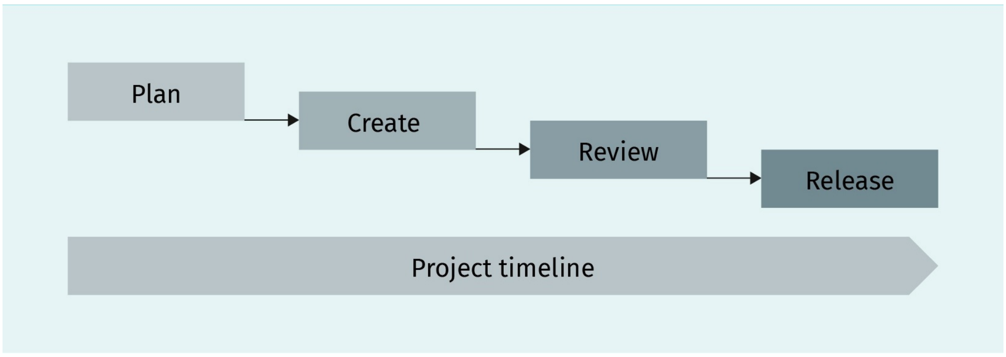
The work of a department or team within a company can be divided into two groups: ongoing operations and new projects. The former deals with day-to-day operations; for example, in a software development department within an automobile manufacturer, the main goal is to create software for the company. The tasks usually carried out by the operations department do not have a specific deadline and are used to solve problems and to achieve similar and consistent results through repeatable procedures or activities. The latter department deals with projects, i.e., specific tasks characterized by a unique outcome and a clear start and end time. Projects cannot be considered a long-term goal for a department because they are usually temporary and their duration is relatively short. However, projects bring innovation and help the company survive in the market. With the advent of innovations, projects are becoming more and more important for companies, which explains the parallel increase in the demand for project management skills. For example, the software development team at an automobile manufacturer may receive a sales and marketing project or create a project aimed at developing a navigation system in cars to help the company compete in the market. To achieve such goals, a company needs project management to carry out all projects inside a team, bring them successfully to completion, and deliver high-quality outcomes. Project management consists of tools, theories, techniques, knowledge, and skills that are used to lead teams, allocate resources, and manage the complexity of a project in order to achieve the prescribed goal with certain success criteria while remaining within a specific time frame and under budget. During the last few decades, different management organizations have tried to introduce a standard that covers all aspects of project management. With the significant industrial growth that started in the 1950s, companies began to systematically apply project management tools and techniques to the development phases of their projects (Kwak, 2005). One of the main standards, which was intensively used until 2000 and is still used by many companies today, is traditional project management (TPM) (Project Management Institute, 2017).

1.1 Traditional Project Management

TPM is a set of techniques and tools that can be applied to an activity or task aimed at creating an end product, reaching a specific outcome, or offering a service. This standard is based on the **waterfall model** presented in the figure below:

Waterfall model
This is a linear sequential development methodology in which each stage or step happens after finishing the previous step (for example, the testing step coming after the implementation step).

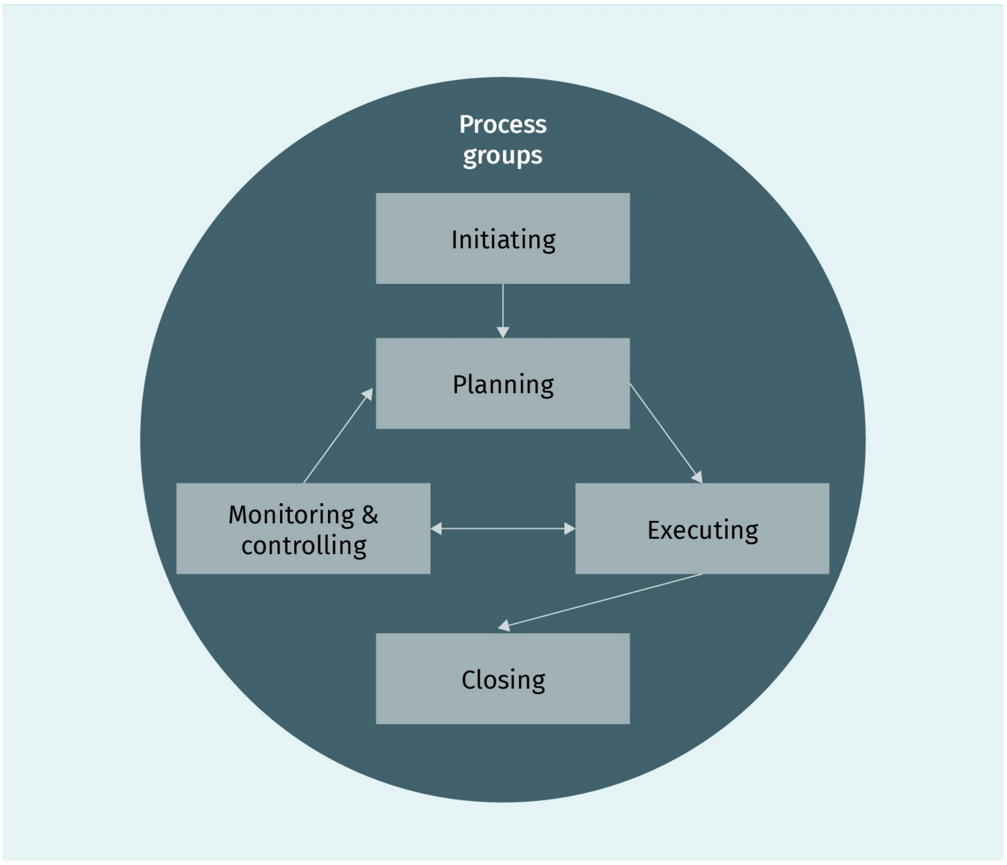
Figure 1: Waterfall Workflow



Source: Asadi (2020).

The main idea behind TPM is to realize projects in a predictable and linear way thanks to the clear definition of the boundaries of each phase, so that the plan can be followed without having to make many alterations (Project Management Institute, 2017). In today's world, some projects have a clear goal and will not have to undergo many changes during their lifetime, which means that their target can be achieved through a linear and sequential cycle of events or phases. This type of project can be perfectly managed by TPM, because this approach has a fixed lifecycle based on the following sequence.

Figure 2: TPM Workflow



Source: Department of Veteran Affairs (2020).

When the idea is initiated, a new project or phase is created within the current project. The management team begins to identify the resources required, determine the scope and goals of the project, and to define how those goals can be achieved. Then, the development team begins execution to carry out the tasks defined in the project plan, while management uses monitoring or control to track and review the progress and performance of each phase. Once all related activities have been completed and the goal has been reached, the manager formally closes the project or one of its phases. In TPM, the project is usually divided into clearly defined stages, each with their own planned targets and outcomes. Before moving from one stage to the next, the target of the previous stage must be achieved, all related tasks must be carried out, and all decisions must be made. TPM puts a special emphasis on linear phases, planning in advance, documentation, and on the prioritization of the defined tasks. For the whole duration of the project, TPM will ensure that it is following the predefined phases. When drafting a complete plan that will cover all requirements and risks for each phase of the project, TPM usually assumes that the environment and resources will remain the same throughout. It breaks down the outcome into smaller tasks, which leads to an increased predictability for each task. Therefore, thanks to a thorough prediction or estimation of the tasks required and the risks associated with them, TPM is able to keep a project under control. Based on such a strategy, the target of

TPM is to increase the efficiency of the initial project plan in order to deliver the outcome within the planned time frame, budget, and scope. The main advantages of the TPM approach are the following:

- It is easy to understand and follow because it is a sequential and controllable process.
- It is cost-efficient because the fixed assumptions made by TPM lead to a reduction in project costs.
- It is efficient because remote projects usually require less communication and a more detailed plan, something TPM is particularly good for when carrying out such projects.
- It increases customer satisfaction or accountability by achieving the targets by the planned deadline.

However, this approach also presents the following disadvantages:

- Since it is based on prediction, its procedures can be considerably slowed and lead to timing issues whenever the actual costs, effort, and resources for the project differ from the initial estimation.
- There is less customer interaction compared with the other methods we will explore in this unit, which can lead to a misunderstanding of the target or outcome of the project.
- It can deprive employees of their creativity because of its predefined and sequential phases.
- It can cause a lack of coordination when each team member is busy with a task and only thinking of their individual effort, talent, and outcome.

Several methods introduce a detailed definition of each step or phase in TPM, as well as tools and techniques that can be used by project managers. Two well-known TPM methods which will be presented in this unit are PMBOK and PRINCE2.

PMBOK

PMBOK (project management body of knowledge) was originally introduced by the Project Management Institute (PMI). Founded in 1969, the PMI is a non-profit organization that provides support and sets the standards for project management across the world (Project Management Institute, 2017). PMBOK was first published in 1983 as a set of detailed guidelines and definitions of standard terminology frequently used in project management (Project Management Institute, 2017). This resource can be used by managers to ensure that projects in different teams and organizations meet the high standards of the PMI. The main assumption of PMBOK is that project managers need a standard that is applicable to any kind of project in any industry and sector regardless of differences in culture. It is comprised of five initial processes and ten knowledge areas. These knowledge areas are related, but not all of them can be applied to every single process. The five processes—initiating, planning, execution, monitoring, and closing—are based on TPM and are common to almost all projects. Each process describes the tasks which need to be carried out in order to manage any project. These are defined as required inputs (documents, plans, designs, etc.), expected outputs (documents, plans, design, etc.), and tools or techniques that can be used to convert inputs into an expected output. The output of a process will then constitute the input of the next process. Each process should cover a specific need of the project, and the management team has to choose which process should be used, by whom, how, and when.

PMBOK also includes lists of best practices, conventions, and techniques which comply with industry standards and enable a team or an organization to manage projects in the best possible way. With best practices in PMBOK, we refer to the general agreement that the application of the knowledge, tools, and techniques in PMBOK can increase the chance of success of different projects (Project Management Institute, 2017). The PMI tries to regularly update the guidelines introduced in PMBOK to ensure they are always based on the most up-to-date project management practices and considers the growing importance of innovations. The current, sixth edition of PMBOK was published in 2017 (Project Management Institute, 2017). PMBOK identifies ten different areas of knowledge that determine what processes must be carried out in order to achieve effective project management. These areas of knowledge are the following:

1. Integration, which defines and combines various processes inside the different project management process groups
2. Scope, which determines the work required to successfully carry out a project
3. Schedule, which ensures that the project will be finished by the scheduled time
4. Cost, which involves controlling, estimating the costs, and funding the project to ensure that it will be completed under the approved budget
5. Quality, which identifies the policies that help deliver a high-quality outcome
6. Resource, which involves organizing and leading the team members of the project and managing the resources available
7. Communications, which involves creating, collecting, distributing, managing, and monitoring the project information
8. Risk, which involves identifying, planning, analyzing, and controlling the risks associated with each stage of the project
9. Procurement, which identifies and purchases the external services and products that are required to finish the project
10. Stakeholder, which involves analyzing and engaging the stockholders, people, or organizations that can have an impact on the project

PMBOK is not a methodology, but rather a set of guidelines and knowledge that govern the lifecycle of a project or program. Any industry or organization can create its own best practices based on its scope by applying the processes illustrated in the PMBOK. However, this method also presents a disadvantage, in that it has so many combinations of processes and knowledge areas that these can add unnecessary complexity to small-scale projects; it is therefore important that they are always adapted to the scope and size of each individual project.

PRINCE2

PRINCE2 (projects in controlled environments) is an approach introduced and released by the UK government in 1990 to support project management both in IT and non-IT industries. Over time, it became a well-known standard and was also adopted by the European Organization (Siegelaub, 2004). Unlike PMBOK, which is a set of guidelines or standards, PRINCE2 is a generic project management framework developed as an industry-independent standard that focuses not only on the process but also on achieving a high-quality product. This difference makes it a perspective and product-based project management model. The inherent product focus of PRINCE2 reflects the reality and target of projects to deliver business needs as an outcome of projects. PRINCE2 is a methodology in its own right and its framework is based on seven principles, seven themes, and seven processes that need to be applied by project management and must be tailored to the needs of each individual project (Siegelaub, 2004). The seven principles of PRINCE2 are

1. continue business justification,
2. learn from experience,
3. define roles and responsibilities,
4. manage by stages,
5. manage by exception,
6. focus on products, and
7. tailor to suit the project environment.

Throughout the project lifecycle, PRINCE2 also identifies seven themes that need to be continually addressed to identify the project's tasks and activities:

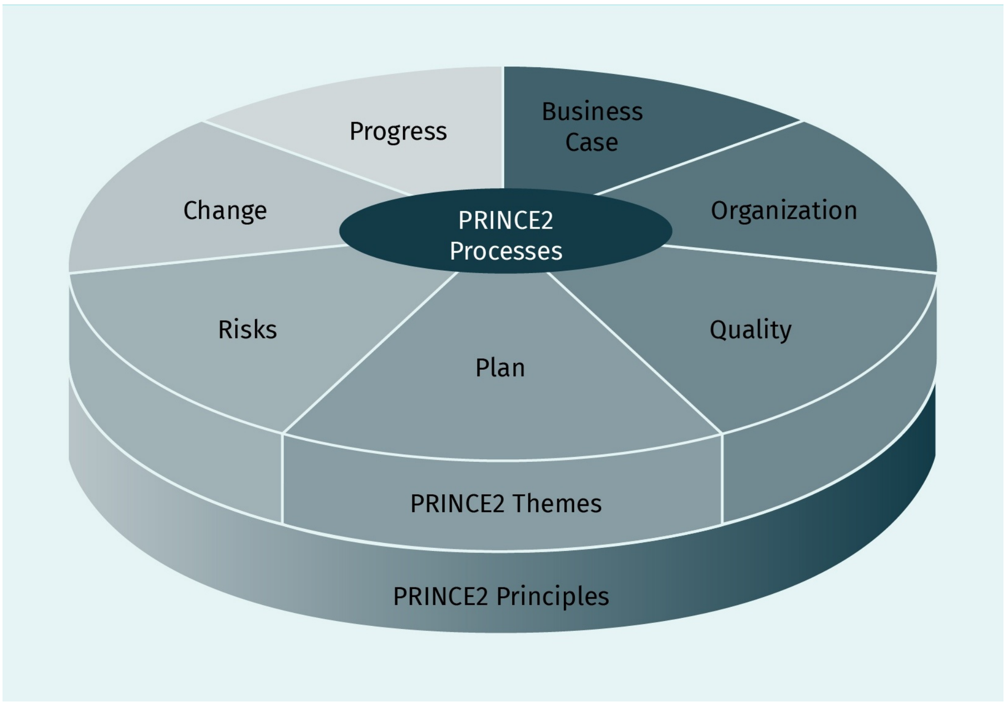
1. Business case, which is based on the principle of continued business justification, which provides knowledge about whether a project is worthwhile and achievable
2. Change, which provides knowledge about how to handle and manage changes that arise during the project
3. Organization, which defines roles and responsibilities so that they can all be on record
4. Plans, which describes how the team can achieve their target to deliver a product under budget and within the specified time frame
5. Progress, which is about tracking the project, allowing the team to check its status at any time
6. Quality, which is related to the focus on products principle and helps to check the quality of each phase
7. Risk, which identifies and controls risks associated with the project

To a certain extent, these themes correspond to the knowledge areas of PMBOK. PRINCE2 also identifies seven processes that should be implemented in order to reach a specific objective. The seven processes are

1. starting a project,
2. initiating a project,
3. directing a project,
4. controlling a stage,
5. managing product delivery,
6. managing stage boundaries, and
7. closing a project.

The structure of PRINCE2 is presented below.

Figure 3: PRINCE2 Structure



Source: Ranjan (2014).

Since different activities have been identified across the seven processes illustrated in PRINCE2, it is necessary to determine who is responsible for each of them and when they will be executed (Bentley, 2010). Once that has been decided, the clear definition of the process in PRINCE2 makes project management an easy feat for people with less experience in this area, and it greatly improves the quality of the project outcome. However, although PRINCE2 has a well-defined workflow, using such an approach does not guarantee the success of projects that require more complex project management. Furthermore, as it focuses on a wide range of different activities, it can also be the wrong approach for a number of smaller projects.

1.2 Agile Project Management

Although during the last few years many efficient project management methodologies have been introduced, there still is no one-size-fits-all method that can be adopted to manage all projects and carry them out successfully. TPM is more effective for projects with low levels of uncertainty; clear requirements with a low change rate; clear targets; and less customer interaction. In the methodologies based on TPM, the main emphasis lies on the initial planning and estimation of each project phase. During the TPM process, it is not necessary to involve customers and end consumers because the main focus will be on the effort to document every activity. For projects where TPM would not be the ideal methodology, **Agile** can be a good alternative. This method originated from software development and was first introduced in the Agile Manifesto, a short document published in 2001 which is comprised of four core values and twelve principles (Fowler & Highsmith, 2001). The four core values of Agile are

Agile
This type of project management is an iterative approach to delivering a project throughout the iterative development loop called a sprint.

1. individuals and interactions over processes and tools. Communication and interaction with customers and other parties is more important than standard processes and tools. It is also important that all people involved in a project or team are working and collaborating effectively.
2. working software over comprehensive documentation. The focus should be more on delivering a working application than on writing comprehensive documentation. The main point of a project is to create a product, not documents.
3. customer collaboration over contract negotiation. Collaborating with customers is more important than signing a contract with them. A contract is important to start the project, but customer requirements may change over time and may not be covered in the initial contract.
4. responding to change over following a plan. Since responding to change is more important than following the plan, flexibility should be an essential element of the process in order to avoid freezing the scope of the project.

As mentioned above, the Agile Manifesto also includes the twelve principles of agile project management (Fowler & Highsmith, 2001). These principles describe the flexible approach Agile has towards changes and its focus on communication with the customers. The Agile principles are explained below.

1. Customer satisfaction: This is achieved through early and continuous delivery of a valuable product. Customers will be happy to receive working software on a regular basis rather than waiting for it to be released.
2. Change: New requirement changes are constantly received and implemented throughout the development process.
3. Delivery: A working software or product should be delivered frequently, preferably on a weekly rather than monthly basis.
4. Collaboration: There should be a close collaboration between the business stakeholders and the developer team during the project lifetime. If the customer is aware and part of the decision-making process, the latter will improve considerably.
5. Relation: The people involved in the project should be motivated, supported, and trusted.
6. Communication: Once the developer teams have been formed, enabling face-to-face communication and interactions becomes the key to the project's success.
7. Measure of progress: The primary measure of progress is working or functional software.
8. Promote: Agile processes promote sustainable development; the customers, developers, or stakeholders should be able to maintain the product.
9. Attention: Continued attention should be paid to technical excellence, detail, and design.
10. Simplicity: The project should have a clear purpose and principles that can be understood by everybody so that they can suggest possible changes.

11. Architecture: Great architectures, requirements, and designs come from self-organizing teams. When people are motivated, they take on more responsibility and they become better decision-makers by communicating with others. This results in a high-quality collaboration and a high-quality product.
12. Feedback: Regular feedback should be provided on how to become more effective; self-improvement, advanced skills, tools, and process improvement help teams to become more efficient.

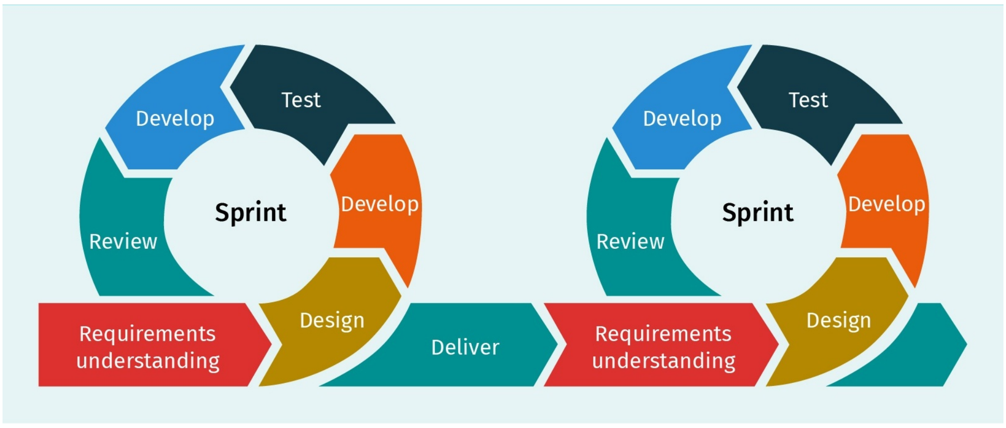
Now that the values and principles of agile project management have been defined, we can go on exploring this new methodology. Agile is an iterative project management methodology that helps organizations deliver products or services quickly and easily to their customers. Its structure corresponds to the different levels of an organization, e.g., staff, team, department (Highsmith, 2009). Depending on how the organization works, the application of the Agile methodology to project development can improve team collaboration and delivery performance. In fact, this method mainly relies on teamwork, communication, collaboration, time-boxing, and quick reaction to the changes that happen during the lifecycle of a project. Agile puts a particular emphasis on the process of software or project development and focuses especially on the quality of processes, software, or projects. In order to make the Agile process deliverable and shorten the lifetime of projects, the following characteristics of Agile need to be considered:

- Iterations. Agile focuses on a single requirement for the whole duration of a short cycle constituted by fast verification, fast correction, and running (using multiple iterations).
- Modularity. During the development process, Agile breaks down a complete system into smaller and more manageable components called modules.
- Time-boxing. When applying the Agile methodology, a team or a developer has a set amount of time in iteration cycles of one to six weeks to complete a task for each module.
- Parsimony. The Agile methodology entails removing all unnecessary activities in order to control risks and reach the target.
- Adaptive. Agile project management adapts quickly to new risks or changes.
- Incremental. Agile project management requires a development of the product in increments, thereby allowing functional application building in small steps. At the end, all increments will be integrated into the complete system or product.
- Convergent. Increment risks are convergent in the Agile process.
- Collaborative. The Agile approach is inherently communicative and collaborative.
- Customer-oriented. Customer satisfaction is the top priority in Agile, favoring people over processes and technology.

Sprint
A Sprint is a short period of time during which a Scrum tries to get a certain amount of work done.

The Agile methodology is based on an iterative process in which projects are broken down into small parts or increments known as **sprints**. The Agile workflow consists of sprints that follow the software development lifecycle, which includes an understanding of requirements, design, development, testing, development, review, and deployment or delivery. An example can be seen in the following figure:

Figure 4: Agile Workflow



Source: Asadi (2020).

Generally speaking, the Agile process means less time planning or prioritizing and more time understanding customer requirements and delivering results. Since customer satisfaction has the highest priority in Agile, customers are directly involved in each iteration and are asked to review the delivered product at the end of each phase or Sprint. Because of its inherent iteration, this method is more flexible and therefore adapts more easily to changes that have to be implemented to satisfy customer requirements. All new requirements are usually considered in the next loop of the iteration. By breaking down the project into small sections, the manager gains more control, which allows him to better manage the risks associated with it and quickly resolve problems when they arise. In the Agile methodology, there are also different roles that must be distributed among the team members involved in Agile-based project development.

- User or customer: An Agile process always begins by considering customers and their requirements.
- Product owner: This role gives voice to a customer or an internal stakeholder, so it entails analyzing and summarizing ideas, gaining knowledge, and giving feedback on a product. While working with the development team, a product owner divides customer requirements into **user stories** and adds more details. These details should cover who the user or customer is, what type of problem has been solved for them, why the new solution is important to them, and what acceptance criteria apply to the solution. A product owner is a team member responsible for defining the vision and working with the development team to create an acceptable solution for it.
- Product or software development team: In the Agile methodology, a team uses its advanced skills to complete a project and focus primarily on the delivery of functional products. In order to do this, team members check in frequently—even daily—with each other to see how far everyone has progressed and check who is responsible for specific tasks.

User story
A user story is a simplified descriptive document used in Agile which describes the type of customer, requirements, and why to create a function for the customer.

Agile Advantages and Disadvantages

The main goal of project management is to get a result on time and under a certain budget. Like any other method, Agile presents some advantages and disadvantages that may affect a team's ability to achieve the main goal in a positive or negative sense, so it is particularly important that teams are aware of this when introducing Agile into their organization. When considering the characteristics and principles in an Agile team, the main advantages for the people and organizations involved in the project are the following:

- Less detailed requirements will be needed for starting the project because further information can be gained during the lifetime of the project (Sharma et al., 2012).
- Good face-to-face communication between developer teams and customers is the main principle of Agile; it will also help to reduce risks associated with development (Sharma et al., 2012).
- An early delivery of the product benefits customers (Masson et al., 2007).
- A product which is developed by an Agile-based development team will have a reduced time-to-market compared to a product developed in a TPM-based development team (Carilli, 2013).
- Agile reduces the costs of projects, which might be higher for projects developed with the TPM approach (Carilli, 2013).
- Accepting changes and involving customers in each step and decision guarantees higher product quality (Masson et al., 2007).

However, Agile is not a perfect model and it cannot automatically guarantee the success of a project. When deciding which approach to adopt when developing a project, the following disadvantages of Agile must also be considered:

- Agile has a less detailed initial planning, which can lead to a wrong estimation of efforts and costs (Serrador & Pinto, 2015).
- Agile has less documentation, which can lead to less information for all present and future developers or new developers (Sharma et al., 2012).
- An Agile-based project needs clear customer feedback and communication in order to be improved. If customers are not representative and are not giving clear feedback, this will slow down the development team considerably (Sharma et al., 2012).
- An Agile team must be formed by experienced developers; it has to be faster and require less time in planning the project, resulting in higher costs (Taibi et al., 2017).

Despite the aforementioned disadvantages, Agile is a very successful methodology and it has helped the IT industry to deliver high-quality products. Nowadays, several well-known companies such as IBM, Microsoft, Apple, and Adobe are using Agile project management to change and update their programs through the so-called Agile transformations (Carilli, 2013). These are major organizational changes that involve Agile working in small multidisciplinary teams that focus on delivering results quickly, experimentally, and iteratively (Consultancy, 2020). For example, in 2012, Adobe switched to Agile in order to decrease the amount of hours spent on managerial activities and they managed to save 80,000 hours annually (Hearn, 2019). Over time, different Agile-based approaches have been introduced, developed, tested, and released in different industries to allow for a quick delivery of successful systems. Two very well-known approaches are Scrum and Kanban.

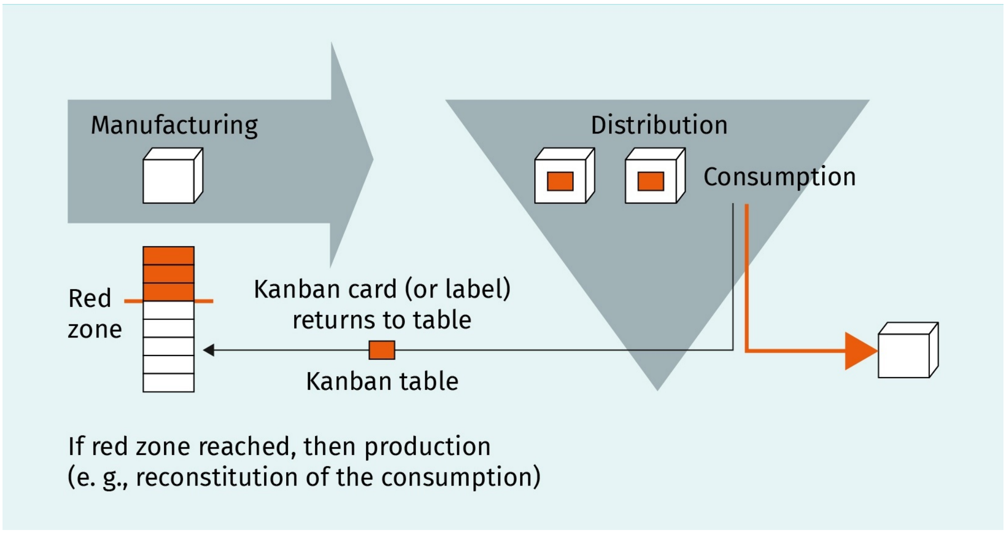
1.3 Kanban

The Kanban method is a visual system invented by Toyota to help organize the workflow in project management. In Japanese, the name Kanban means “sign” or “board” (Sugimori et al., 1977). Previously, most industries worked based on a push system, which means production was not based on actual customer demand; rather, manufacturers delivered their products according to estimates based on previous demand, without considering the current state of the market or customer requests. Instead, Kanban is a **pull system** that is part of the Agile movement. A pull system is based on actual customer requests and sends a task to the production line only when there is the demand for a certain product. Kanban has been adapted and applied to various industries, where it has helped to minimize activity without affecting productivity and to create more value for customers while keeping costs low for the industries. This visual model focuses on tasks and shows what to do, when, and how much. This method is based on three main parts: the Kanban board, cards, and swim-lanes.

Pull system
This technique is used to start a new task when there is a customer request, which optimizes costs and reduces waste in production processes.

The original Kanban system is presented in the following figure:

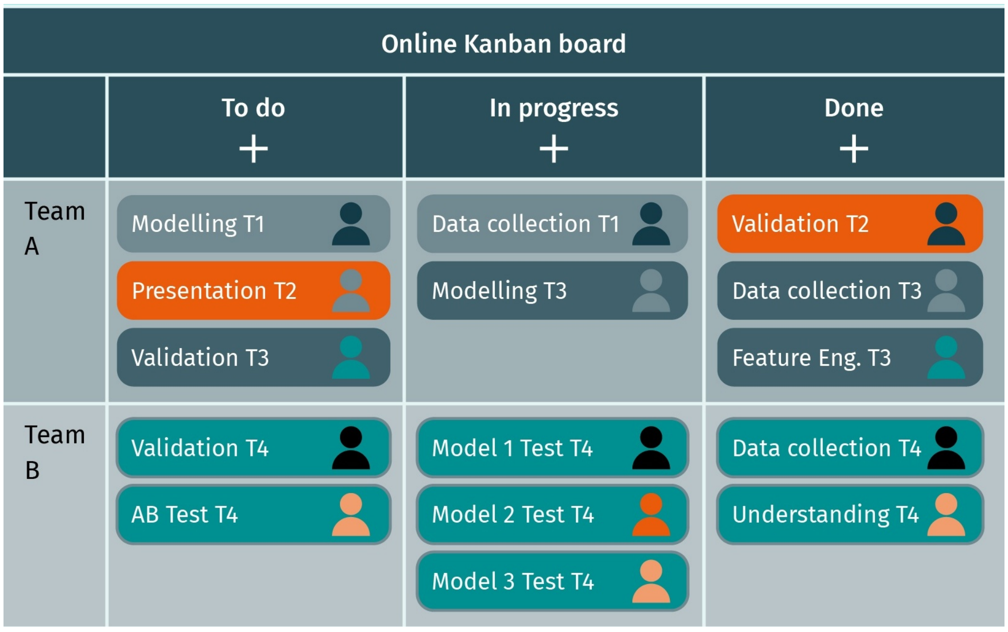
Figure 5: Kanban Structure


Manufacturing
Distribution
Consumption
Red zone
Kanban card (or label) returns to table
Kanban table
If red zone reached, then production
(e. g., reconstitution of the consumption)

Source: Waldner (1992).

According to this structure, a Kanban card is pulled from the red zone whenever there is a consumption signal, and a task will be started simultaneously in the production line. After production is complete, the product is delivered and the card is returned to the Kanban table. The board that shows all the tasks and their progress is called a Kanban board. The Kanban board has different columns showing the status of tasks, and the number of columns changes based on the structure of the team and project. However, three main columns corresponding to generated ideas, work in progress, and completed tasks are always present on the board. These columns can be marked as “to do,” “in progress,” or “done.” The card you use to view a task is called a Kanban card and it shows your team the progress and status of a specific task. If you have different categories of tasks, you can use Kanban swim-lanes to separate those categories horizontally on the board. An example of the Kanban board is shown in the picture below:

Figure 6: Online Kanban Board



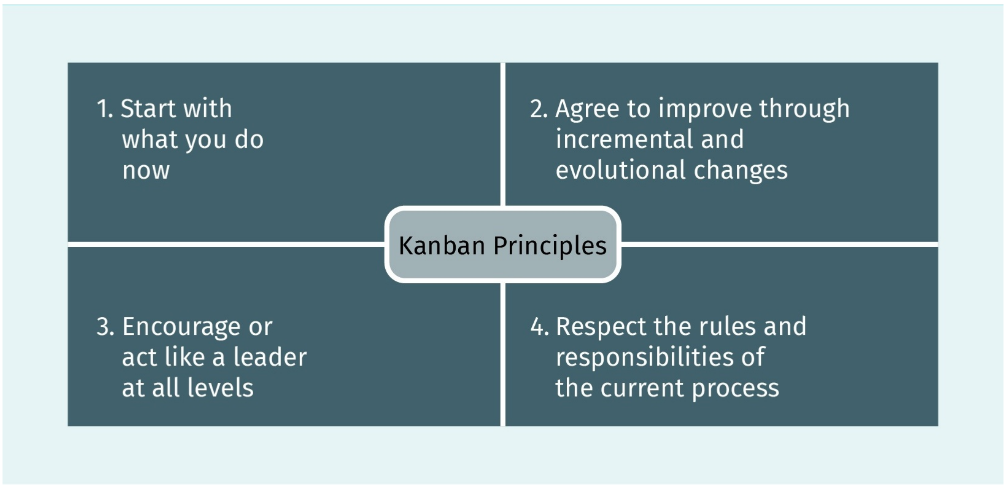
Source: Asadi (2020).

With the increase of technological progress and the growth of internet usage, many online platforms also offer Kanban nowadays. However, despite the availability of online tools, many companies still follow the traditional Kanban philosophy that advocates for the use of physical boards to improve interaction (Collado, 2018). The Kanban board helps you and your team to maintain transparency regarding the status of the work; it also allows you to quickly identify bottlenecks and remove any obstacles and risks associated with the project.

Principles and General Practices

To successfully develop the Kanban method, Toyota introduced four core principles and six general practices that a team should consider when using Kanban (Sugimori et al., 1977). The four principles are illustrated in the figure below:

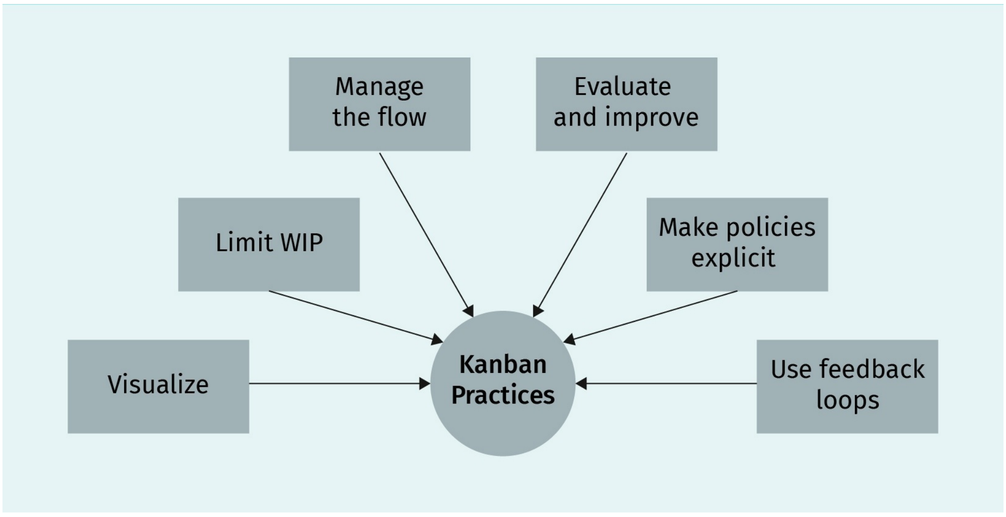
Figure 7: Kanban Principles


1. Start with what you do now
2. Agree to improve through incremental and evolutional changes
Kanban Principles
3. Encourage or act like a leader at all levels
4. Respect the rules and responsibilities of the current process

Source: Asadi (2020).

These principles are essential to improve collaboration in the Kanban process and to encourage respect within a team. As mentioned before, the Kanban method also includes six practices that should be applied when starting a new Kanban cycle which are illustrated in the figure below:

Figure 8: Kanban Practices



Source: Asadi (2020).

As explained earlier, Kanban focuses on visualizing tasks by adding them to the Kanban board. Each task should have a name and can also contain additional information such as: categories, creation date, deadline, owner, or requirements. When adding a task on the Kanban board, the team needs to limit the number of cards marked “in progress” to increase productivity and focus. Each task will be executed during a sprint and, if it is not possible to do so in the current increment, it can be moved to the next one. When a team first starts using Kanban, it will be difficult to set a perfect limit of tasks “in progress,” so it is advisable to start with a low or manageable limit as it will achieve faster results. Over time, the team can increase this limit if they feel comfortable trying to carry out all tasks on the column marked “in progress.” If, during the current sprint, the team members feel that there is time to work on another ticket, then they can pull a task and start executing it without waiting for a new sprint. When moving tasks on the Kanban board, it is important to manage the flow to recognize who is doing what and when. It is important to identify this in order to successfully manage the work of the team. Kanban, in fact, is not about micro-managing people and overloading them with tasks, but rather about providing a transparent, healthy, and productive system that will allow a team to execute tasks faster. For this reason, making policies explicit will help to move in this direction, as well as to connect people and work. An interesting thing about Kanban is that team members can give and receive feedback that will help the team be more agile. The team can have a short daily stand-up meeting (about 15 minutes or less) held in front of the Kanban board to provide updates and feedback. To improve the quality of the tasks or their outcome, a review can be carried out by another team member to make sure that the outcome meets the team’s standards and to provide feedback to the owner of the task.

Benefits of Kanban

Introducing Kanban in a team doesn’t actually require revolutionary changes because its concepts, implementation, and procedures are easily understandable even for people without a project management background. By using an Agile-based method like Kanban, the team can get more work done and always be aware of all the tasks that are being executed or are still waiting. Since all tasks can be viewed on the board, this constitutes an information hub for the team and helps to obtain all information needed to execute a specific task. When moving a task to a column it is easy to see whether it is already overcrowded, which will help the team to recognize the bottlenecks in the workflow. Therefore, thanks to Kanban’s limited “work in progress” policy, the team will be more responsive and productive, as well as satisfied with their workload. Using Kanban then will certainly bring more flexibility, as changes—like adding a new task—can be made directly on the board and will be prioritized and solved by the next team member available. The quick response to changes makes your Agile team more productive and will improve collaboration, since all team members will be able to support each other and will have the capacity to execute each other’s tasks. The main benefits of Kanban can therefore be summarized as follows:

- Kanban is easy to understand and to implement.
- The team can receive information about tasks and their status quickly from a visual Kanban board.
- All changes can be added as new tasks to the “to do” list on the Kanban board, which will enable the team to adapt quickly to them.

Agile Kanban

Agile Kanban is a combination of the principles of the Agile project management methodology and Kanban. In this method, Kanban is used as a part of Agile to manage process workflow and bring user stories to completion. Each user story or task in Agile corresponds to a card on the Kanban board and is moved from "to do," through "in progress," and finally to "done" after careful consideration of all policies and after visualizing each necessary step. The Agile team leader should set the limit for the tasks "in progress" based on the team's capacity to be more productive, focused, and flexible during the development cycle. Tasks are prioritized based on customer requirements and changes are accepted and considered in the workflow. The general focus is on delivering an outcome and bring value to customers faster thanks to successful collaboration among team members. Kanban can be used in any industry or project where the necessary tasks can be predicted and where a pull system is available. If this is the case, Kanban fits the organization perfectly. For example, suppose you work on the data science team of an automotive company where you have all data ready and accessible from the start and the technical department regularly asks your team questions on how to improve their product that need to be answered based on the data sources. Each of these questions will constitute a task on the Kanban board; once you have met all requirements and finished your current task, you can pull a new one from the "to do" column of the Kanban board, thereby answering all questions from the technical department one by one. Although Kanban can be the perfect method for organizations based on a pull system, it might be wiser to choose a different approach for organizations based on a push system dealing with more complex projects and many stakeholders. For example, a retail company is a push-based system. When they have a new product, they launch it based on previous customer demand for other products and then they use marketing to promote it.

1.4 Scrum

Scrum is a project management method first introduced in the Harvard Business Review by Hirotaka Takeuchi and Ikujiro Nonaka. In their article, the authors use a rugby game as a metaphor to describe the benefits of a structured and self-organized team and how to make it innovative and productive (Takeuchi & Nonaka, 1986). Scrum is a term borrowed from rugby and refers to the way in which a team forms a circle in order to gain possession of the ball. An example of this can be seen in the following figure:

Figure 9: Scrum in Rugby

Source: Baucherel (2019).

Agile Scrum is a method first introduced in software development by Schwaber and Babatunde at the DuPont Research Station and the University of Delaware, where it was used to lead a software development team and then applied to the Easel Software Development Corporation (Sutherland, 2004). This method entails taking development teams through iteration loops in order to achieve incremental and product results. It provides a strategy that a team can follow to learn how a product works, leverage experience, gain knowledge, and make adjustments based on the changes implemented in order to create and deliver high-quality results. Furthermore, the team members can organize themselves and improve the quality of work by communicating regularly with each other and with other shareholders and by working together to solve problems as they arise. This model is best suited for companies or teams whose product development processes can be divided into two- to four-weeks-long iterations.

Agile Scrum Frameworks

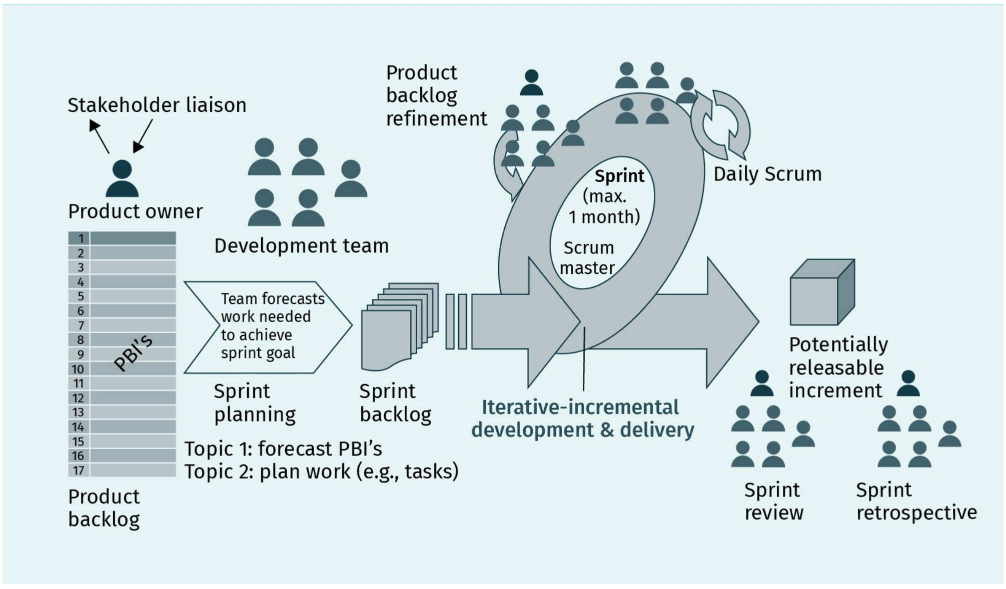
To create a successful Agile Scrum framework, team members need to consider some original principles (Takeuchi & Nonaka, 1986).

- Transparency: The work environment and culture should enable the team to share their knowledge about the process and the obstacles associated with it in order to solve problems as they arise.
- Inspection: In order to improve the quality of the final output and understand how the process works, the team must have review tasks regularly and all team members should be open to feedback.

- Adaptation: The team should constantly check and remove the items that do not add value to the project.

The different elements and roles that constitute a successful Scrum framework are shown in the figure below:

Figure 10: Agile Scrum Framework



Source: Mitchell (2015).

Sprint in Agile Scrum

In Agile Scrum, the project and its scope can change, but the cost and time necessary to reach the outcome remain unvaried. The project length is broken down into equal parts—or sprints—based on the development effort and time required to bring them to completion. Similarly to the Agile sprint, in Agile Scrum, a sprint can become a project in itself at the end of which a version of a product can be introduced to customers. The time needed to complete a sprint should remain constant in order to improve the accuracy of the cost estimate; ideally, a sprint should always be one or two weeks long. For example, with a sprint duration of two weeks the costs for the people involved in the project will be roughly half of a monthly salary.

Roles in Agile Scrum

In Agile Scrum there are various roles that team members can adopt based on their experience and that are essential for a successful Agile team (Sims & Johnson, 2012). These roles include

- product owner, which is basically the same role as an Agile product owner. They are responsible for managing the product backlog to reach the desired outcome and benefit customers.

- Scrum master, who is responsible for ensuring that the team follows Agile values and principles and ensures that the sprint goal will be achieved at the end of the sprint. The Scrum master does not directly lead the team, but rather focuses on them, encourages them, and ensures that they have all the requirements necessary to successfully execute the tasks. During the sprint, the Scrum master is the face of the team and works closely with the product owner.
- developer, which is a perfect role for a team member passionate about product development, who also possesses the skills needed to deliver a high-quality product quickly and satisfy customers.

Artifacts as Essential Elements of Agile Scrum

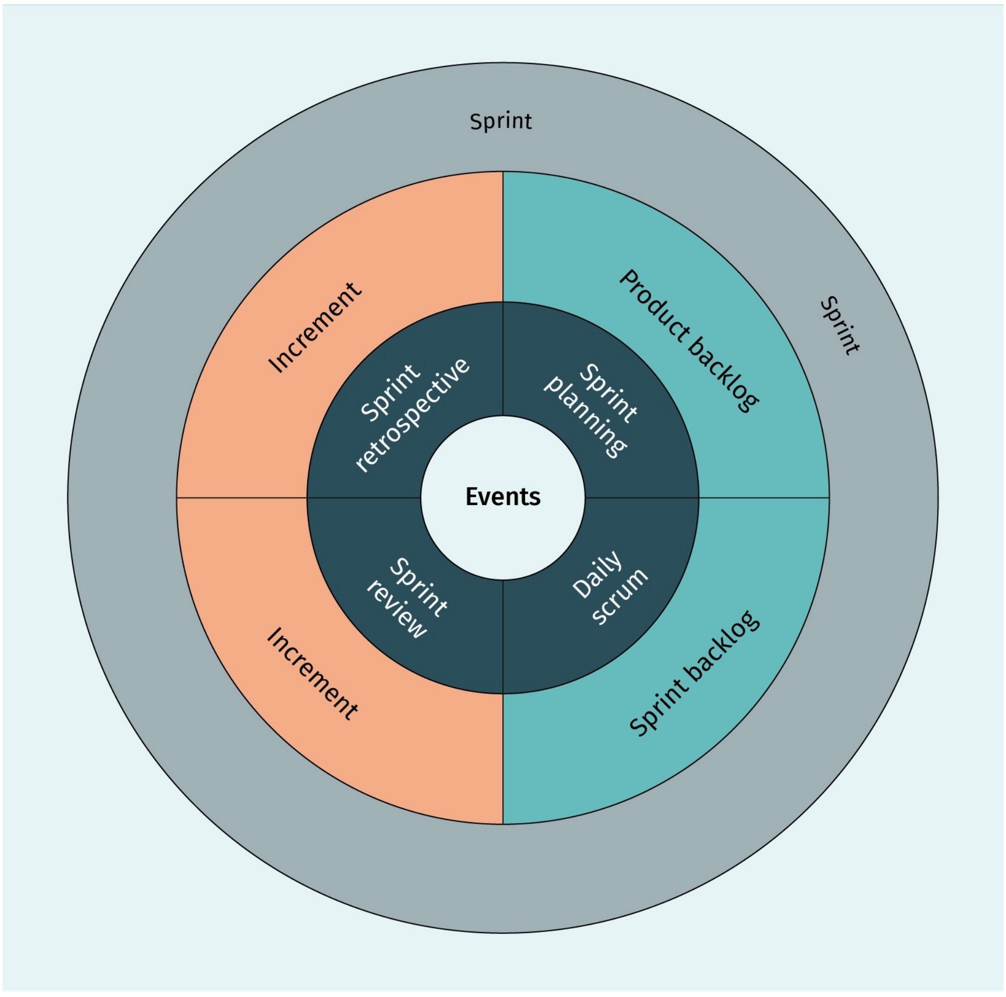
Once the available roles in Agile Scrum have been defined, it is also important to note that the Agile Scrum framework is based on some specific tools called artifacts (Jongerius et al., 2013), specifically:

- product backlog, which is an ordered list of the requirements needed to make changes to the project. The term used to define each item in the product backlog is user stories. The maintenance of the product backlog is the responsibility of the product owner, who also has to identify the business value of each item. Items in the product backlog are options that the development team can choose from and work on in the next sprint, however, there is no guarantee that all items in the product backlog will be delivered.
- sprint backlog:, which contains a list of items that the development team should work on and helps identify the goal of the sprint. The element or user stories can also be divided into smaller tasks, in which each user story is given an owner. The owner of the user story can also be supported by another team member who can help them to execute a task in the user story. The development team should have a good estimate of what items they can deliver at the end of the sprint because after a sprint has started, it is not possible for the product owner to add an item to the sprint backlog.
- increment, which is a collection of product elements that can potentially represent a reliable and functional output of the sprint. If completed, it is considered “done” and ready to be delivered to the customer.
- "definition of done," which is a set of standards and rules agreed upon by the team that are used to add items to the sprint backlog and get them done by the end of a sprint.

Events in Agile Scrum

During a great sprint cycle there are some major events that need to take place. These events and their relationship to the Agile artifacts are illustrated in the figure below:

Figure 11: Events in Agile Scrum



Source: Asadi (2020).

- Sprint planning: This is a meeting held at the beginning of each sprint to create a plan and identify which tasks will be carried out in the sprint. This meeting normally lasts more than one hour and allows the team to select items from the product backlog and push them to the sprint backlog. Following this step, the product owner, with the agreement of the team, identifies and suggests what items to choose. The development team then set the acceptance criteria needed to reach the outcome and deliver the product.
- Sprint review: At the end of the sprint the entire team reviews all the tasks that have been executed in the current sprint backlog. Stakeholders and customers that are involved in the project can participate in this meeting in order to get an update on the product, give feedback to the developer team, or make changes to the product. These changes can then be added as a new item to the backlog for future sprints.

- Sprint retrospective: At the end of the sprint review, the development team and the product owner can talk about the sprint, positive and negative elements of it, or about which standards or increments need to be changed. This meeting will help to understand the concerns of the developer team and to apply its feedback to the future sprints in order to make them more productive.

Agile Scrum Benefits and Limitations

Agile Scrum is particularly beneficial for industries that have complex project processes, that need to evaluate results, and that care about customers (Jongerius et al., 2013). Its main contribution has been to the software development sector, where it proved to be especially effective for small project development teams (max. 10 people). However, many other industries such as financial services, engineering, and construction also use Agile Scrum because this method is intended to provide adaptable and flexible project processes that may be subject to changes during the development process (Jongerius et al., 2013). Although Agile Scrum succeeds in bringing commitment, courage, focus, and respect to every organization that adopts it, it also presents some limitations. In fact, it is not the ideal approach for teams that work remotely or part-time, teams that need special skills to carry out their project or that have many stakeholders or external dependencies. Having said that, it is then up to the team to decide whether they want to use Scrum or not, since this method can always be optimized to suit the individual needs of each team, which helps to overcome its limitations.

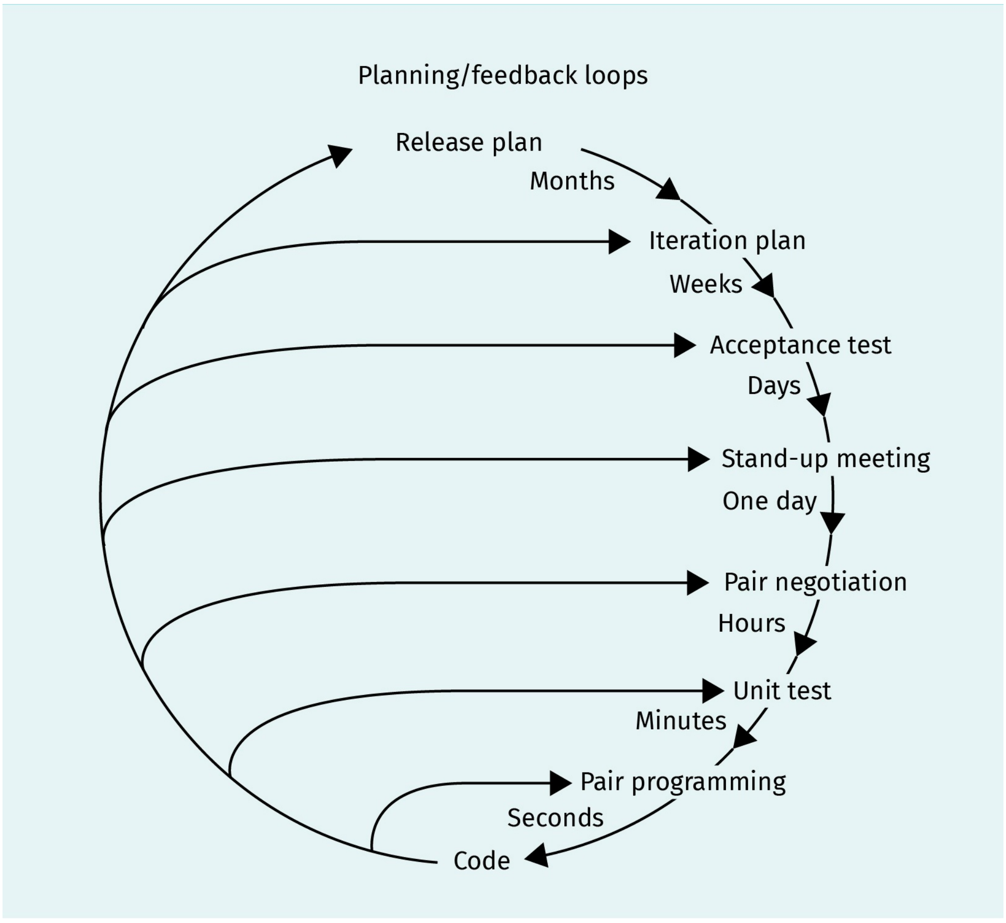
1.5 Other Modern Methodologies

For decades, traditional or Agile management models have created a basis to support and help organizations to meet their project management needs. However, scaling these methods beyond the individual team requires a good understanding of the company's resources, requirements, and skills in order to complete projects and achieve good results. Since the main goal of project management is to complete high-quality projects within a certain time frame and budget constraints, it might happen that one approach works better than another. In order to make the right decision when it comes to choosing a project management approach, teams need a good overview of the different methodologies available and of their advantages. A more detailed explanation of some professional project management approaches is offered in following section.

Extreme Programming (XP)

This Agile-based model was developed by a software engineer at Chrysler while working on a payroll project. It aims to improve the quality of a project by quickly adapting to change (Rosenberg & Stephens, 2008). This model can especially bring great results and create a collaborative environment in projects that undergo shifts in the requirements and that need to receive continuous feedback. The XP model is based on four basic activities: extensive coding, unit testing, listening to the customers, and release. Using these principles in an iteration loop during the project development process allows the developer team to change the scope and do a rapid revision of the project. The iteration structure of these principles is shown below:

Figure 12: Extreme Programming



Source: DonWells (2013).

As you can see, updates from customers are received during a short development cycle, allowing the team to change the scope of the project based on the feedback they receive. The success of this model is the main reason why it has been implemented in many software development companies and industries (Newkirk & Martin, 2000).

Adaptive Project Framework (APF)

Traditional project management methods do not adapt to changes and customer feedback easily. In order to do that, an organization needs to adjust their project management methodology across processes and phases with a solid project plan, something which is impossible in TPM models. Nowadays, global change is inevitable and businesses need to adapt their project management methodology to the project goals in order to remain successful. Using APF is the perfect way to account for unknown factors and changes that may occur during a project phase; this method allows a team to document project functions, sub-functions, requirements, and characteristics before determining the project results. This means that the development team works following a communication cycle rather than sprints. In each cycle, customers or stakeholders can make changes to the project scope which will be considered by the developer and, if approved, implemented in the new phase. Although this closer collaboration enables customers and stakeholders to work closely with the team and determine the success of the project, it can also result in a loss of focus on the development phase.

Six Sigma

In recent years, many companies have tried to improve the capacity of their business processes based on this method, which was developed by an engineer at Motorola to introduce a data-driven methodology in project management (Mehrjerdi, 2011). The main philosophy of this model is defined as measuring, analyzing, improving, and controlling which processes are applied to the input to achieve a high-quality output. By controlling the input, this model helps the organization to successfully deliver a project by using a predictable process, thereby eliminating the root causes of failure during project development. This leads to a reduction in the variation of the process and the achievement of consistent results at a low cost. However, this method requires a successful structure, commitment, and skills, from top management to the developer level; a lack of knowledge and responsibility can have a huge impact on the method output. The only disadvantage is that, since controlling is an essential part of this method, users may feel like they are working in a micromanaged environment.

Scrumban

As Kanban grew, a new method called Scrumban was developed to help the Scrum-based teams explore and reap the benefits of Kanban (Nikitina et al., 2012). The structure of this model looks like Scrum, but the work culture is based on Kanban. In Scrumban, the team uses a visual Kanban board and organizes a small iteration similar to a Scrum sprint, which helps the team to adapt quickly to changes in the project. This model's winning combination of the agility of Scrum and the pull system of Kanban greatly improves processes and results. The process used in Agile Scrum also takes place in Scrumban in order to help the team be more collaborative and be aware of the overall status of the project. The Scrumban equivalent of sprint planning in Agile Scrum is to fill the available capacity by adding cards to the Kanban board, thereby adding a "work in progress" visualization to Scrum in order to improve the productivity of team members. This added benefit is what makes this method particularly suited to an organization that needs to maintain an ongoing project or that has a team that struggles with Scrum and needs more flexibility.

Scrum of Scrum

This is a scaled model of Scrum used to coordinate different business areas with multiple product lines and to connect teams in order to deliver complex solutions. It is used in various organizations and industries such as software development (Mundra et al., 2013). This method brings together high-performing Scrum teams and allows them to work together to achieve project goals, encourages respect and trust among team members, and enables them to align themselves on project processes. For this purpose, the model suggests having teams with a maximum of six members; a larger team size can make communication and deployment particularly difficult, whereas dividing a large team into smaller ones helps to create a well-defined structure and achieve better results. In the structure created with this method, each small team is connected to a core activity called organization delivery through their product owner. Product owners are therefore responsible for communicating and sharing demand and scope between the teams. The relationships between the product owners of the different teams are managed by the chief product owner, who guides them to the product vision (Mundra et al., 2013). Product owners will have daily stand-up meetings to discuss their sprint goals, struggles, and improvements to other projects that their team is responsible for.

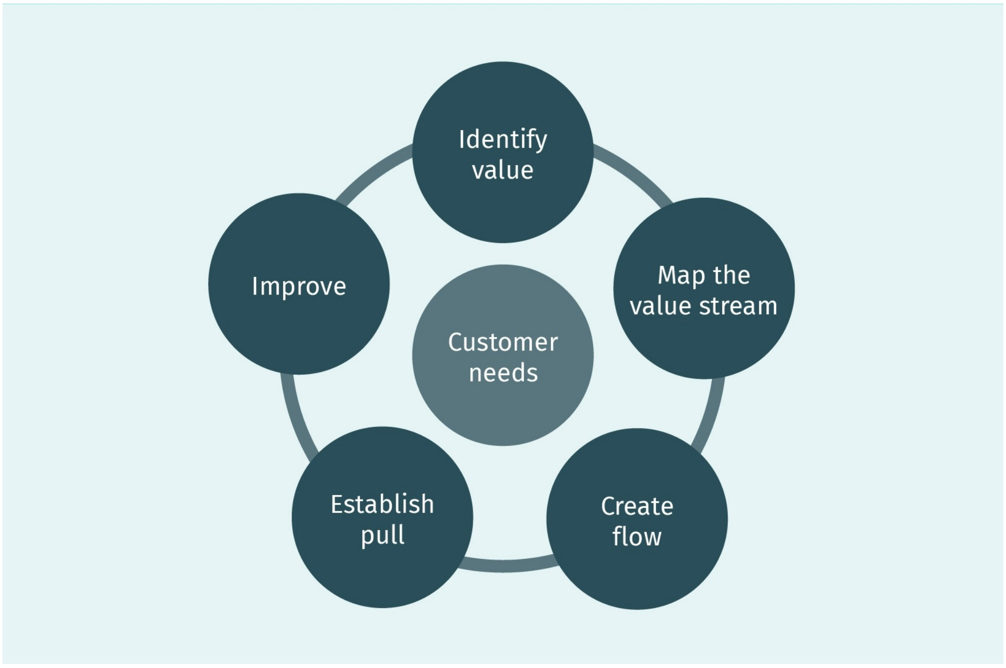
Lean Management

Lean management is a growing topic in the field of project management because this method can be applied to any business or production concept. Companies using this model have experienced an increase in performance and positive value (Ballard & Howell, 2003). The model has three main rules:

- Create added value for customers.
- Eliminate waste or processes that add no value to the end product and do not benefit the customers.
- Continuously improve the product and processes.

This method is not about managing the project development processes, but rather respecting the client, improving the work process, and achieving the planned goals. It also encourages team members to share responsibility and team leaders to share leadership. Based on this method, companies have reduced processes and costs and increased their success in achieving project goals (Ballard & Howell, 2003). In the digital world, lean software development and lean start-up methods were developed based on lean management principles. From a business perspective, lean management shortens the project development cycle and helps determine the business's value; it is based on the five principles shown in the figure below.

Figure 13: Principles of Lean Management



Source: Asadi (2020).

First, it is necessary to identify value by assessing customers' needs. Customers' value will then be used to map the company value stream and create the workflow. To create a truly winning workflow, it is necessary to have a pull system in which work is only pulled when there is a demand for it. This helps to optimize resources in the organization: for example, in a pizza restaurant, a pizza will only be made when there is a demand for it. To reach a great outcome through a stable workflow, the team needs to improve it based on customer demand. Lean management provides some useful guidelines for building a stable team and helps to identify customer changes and quickly remove any obstacles.

1.6 Moving to Agile

The rapid growth of the technology sector and the market can sometimes lead to changes in a company's infrastructure and in the strategy used to adapt to market requests. For some companies, it is not easy to change their traditional structure as well as their project management strategy; for example, if a hardware or auto manufacturer wants to start a software development department to create its own applications, the company might encounter a challenge when switching from traditional project management to Agile. The traditional method gives a company stability, a firm forecast plan, and cost estimates, while the Agile method requires more flexibility and presents some risks associated with its high change rate. However, if the company wants to stay competitive in the market, they might need to switch to the Agile methodology and find a model that suits their needs. A comparison between Agile and traditional project management is illustrated in the table below:

Table 1: Comparison of Agile and Traditional Project Management Methods

	Traditional project management	Agile project management
Requirements	Clear requirements with low change	Clear requirements with high changes
Customer	Not involved in the process	Close collaboration
Documentation	Formal documentation required	Implicit knowledge
Scale of project	Large-scale	Small and medium
Organization structure	Linear	Iterative
Model preferences	Adaptation to changes	Anticipation of changes

Source: Asadi (2020).

If projects are falling behind schedule and the results are not up to the standards and acceptance criteria of the team or customers, it is time to take a closer look at the company's project management approach to see if it might be the cause of such failure. In fact, traditional project management methods can be unnecessarily complex and prove ineffective. Adapting to a newer and better framework like Agile could potentially successfully transform the team. The best model for a new data analytics or science team could be one that is a mix of different Agile approaches, like Scrumban. Using this kind of methodology means that the team can start customizing changes right after receiving feedback from customers. However, making changes can still take time because the team might need more data or customers may need to find the right time to share their business knowledge. This is different in the software development field, where implementing change is relatively easy because the team can change an entire application and introduce new methods or modules without customer input. Regardless of their individual needs, companies should remember that the key to success is to adopt a model that will satisfy both the team and the customers. A sudden move from traditional to agile project management can be particularly tough for a team, especially for members who are not familiar with this methodology and therefore need to learn a whole new project management approach. The company will need to spend time and money to help the team adapt to the new approach. Furthermore, the company will have to change some rules, events, and responsibilities in order to bring in a new language and working culture. On the whole, the smooth alignment of the company's legacy project management method with an Agile framework entails acquiring many new skills and having great patience, but despite the effort, it's definitely worth a try.

SUMMARY

This unit attempted to provide an overview of the many different project management approaches used to develop high-quality products. We began by illustrating the principles of traditional project management

(TPM). This model is easy to understand, particularly suitable for projects carried out locally or remotely, and increases customer satisfaction by achieving the project's goals. However, TPM is also linear and presents some timing issues, it doesn't accept customer changes, and it robs team members of their creativity by forcing them to adhere to a fixed plan. Afterward, we discussed the agile project management framework and its roles, principles, and characteristics. The Agile method has a short development phase and needs fewer details to start a project, entails establishing effective communication between developers and customers, and makes development teams more productive. Next, we explored the Kanban methodology, which uses Kanban boards to help a team visualize tasks. Kanban has a simple workflow based on accepting changes and customer requests, which saves the development team a significant amount of time. We were also introduced to Scrum, another important Agile-based model. Scrum is based on an iterative development process called sprint, where all tasks are distributed among three main roles: product owner, scrum master, and developer. This model can help a team become more productive and implement customer changes at the end of each sprint. We went on to describe the philosophy, advantages, and disadvantages of methods such as extreme programming, adaptive project framework, Six Sigma, Scrumban, Scrum of Scrum, and lean management. Finally, we assessed the advantages of switching to Agile-based methods, among which are improved cooperation, great benefits for small and medium-sized projects, and increased customer satisfaction thanks to the ability to consider customer change during the iterative development phase.

UNIT 2

DEVOPS

STUDY GOALS

On completion of this unit, you will have learned …

- the concept of DevOps and modern lifecycle management.
- the difference between DevOps and traditional lifecycle management.
- what the goals of DevOps are.
- what the impact of DevOps is on teams and development structure.
- how to build a DevOps infrastructure.
- how to build scalable environments.

2. DEVOPS

Introduction

Digital business models, processes, and smart services are largely based on IT and software. Given the agility of today's world, requirements and conditions change very frequently and at a very short notice. In this context, the close cooperation and integration of software development and IT operations with business is becoming an increasingly important competitive factor.

The term DevOps is composed of development (Dev), representing software developers, and operations (Ops), which represents IT operations. The combination of both should enable process improvement in the areas of software development and system administration. Supporters of this corporate culture value collaboration, willingness to experiment, and willingness to learn. More precisely, the aim is to achieve more effective and efficient cooperation between three different departments: development, operations, and quality assurance. These departments pursue the goal of a rapid implementation of stable, high-quality software, starting with the concept and ending with the customer or user (Amazon, n.d.-a). DevOps is not a tool, software, or technology; it is neither a methodology nor a process. DevOps is understood as a corporate culture with certain principles that a company strives to adopt and follow in the long term.

Before diving deeper into the topic of DevOps, it would be helpful to take a look at the concept of application lifecycle management, which is closely related to the concept of DevOps. The term application lifecycle management (ALM) is used to describe lifecycle management from the creation of an idea, through its implementation, to the solution. Through an iterative cycle, a continuous improvement in employee interaction, roles, processes, and information is achieved. ALM consists of different disciplines that were often separated in the context of previously used development processes such as project management, requirements management, software development, testing, quality assurance, deployment, and maintenance. ALM supports the realization of DevOps by integrating all these disciplines, making the collaboration of teams in the organization much more efficient (Chappell, 2008). Generally speaking, there are two main approaches to the ALM: traditional and DevOps. The main differences between traditional and DevOps lifecycle management will be explored in the following sections.

2.1 Traditional Lifecycle Management

In traditional lifecycle management, the roles within an organization operate separately from each other (Shaikh, 2017). Developers have a mindset where change is at the very core of their activities. In order to respond to demanding changes, a company needs developers, who are usually encouraged to innovate and create as much change as possible. Operations, however, do not like change. The company relies on operations to keep things running smoothly and deliver the services that generate money. Operations is justifiably against any kind of change because it can undermine the stability and reliability of the company.

Developers

Developers write software and debug applications. After deploying the application in the production environment, developers are generally interested in its performance and gather feedback from users to make changes and updates to improve the software. The goal of a developer is to continuously create new features and modify existing ones in order to improve them. In other words, the main task of a developer is to implement change in the environment on a regular basis. The role of a traditional developer can be measured by the ability to bring about change. The value of the developer to the organization is typically a reflection of the developer's initiative and ability to generate new applications and innovative features that help users to be more productive.

Operations

The main goal of IT operations or IT administrators is to ensure that everything works optimally. Operations ensure the availability and functionality of hardware and software. Monitoring data traffic, troubleshooting systems and databases, and maintaining infrastructure as smoothly as possible are the central tasks an IT administrator deals with. The role of a traditional operations specialist can be defined by the ability to provide a stable and optimized infrastructure. That means, in essence, implementing as little change as possible to ensure that the infrastructure will remain stable and efficient for the benefit of its users.

Confrontation

Essentially, both developers and operations share one common goal: to make and keep the organization as productive as possible. Despite this fact, it is evident that their roles are at odds with each other and their cooperation is a paradox by definition, because the development and operations teams have different objectives. The development team is trying to build and upgrade software as fast as possible, while the operations team is doing what it can to prevent environmental changes from happening. For this reason, DevOps practices are designed to eliminate the back and forth between these two teams and solve the paradox by introducing new communication standards and promoting a closer collaboration, smoother integration, and the automation of low value processes.

Lifecycle Management with DevOps

Modern lifecycle management can support approaches to agile development by integrating all disciplines, making the collaboration of teams in the organization much more efficient. To this end, the introduction of DevOps enables the continuous delivery of software and updates with frequent releases—sometimes several a day—while completely new releases are only made every few months or once a year. Lifecycle management with DevOps also provides the framework for software development and makes continuous management of the software even easier. Its practices consist of a concise, ready-made plan and of the requirements that turn an idea into an application. When developing software using DevOps principles, the entire lifecycle of the application must be contemplated. Maintenance and future updates must be considered, as well as when the application should be taken out of service and replaced. By combining these factors, DevOps lifecycle management accelerates deployment, improves workflow visibility, delivers higher quality products, and increases developer satisfaction (Golden, 2014). Through the consistent implementation of application lifecycle management by means of **continuous integration (CI)** and **continuous delivery (CD)**, shortcomings in the creation chain of a solution are detected very early on (the concepts of CI and CD will be discussed in more detail in the following section). Through automated quality assurance and distribution of the respective product artifacts, the quality as well as the sustainability of the solution is kept high. Under the term DevOps, development and operations are combined at the same time, which brings great benefits and improvement to an organization.

Continuous integration (CI)
This term refers to the practice of merging all developers' working copies to a shared mainline several times a day.

Continuous delivery (CD)
This is an approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.

2.2 Bringing Development and Operations Together

Development and operations should work better together—the magic word is DevOps, which allows agile working methods to reach the company's everyday life. Nowadays, applications are increasingly developed in an agile manner (AOE, n.d.). Still, it often takes several days or weeks before these applications are put into production by the internal quality and operational processes because deployment is delayed not only by the tools used, but also by the processes behind them. The automation of important IT processes increases effectiveness and efficiency. However, the higher transparency and feedback possibilities achieved in this way often reveal new weaknesses. Yet, all this must not necessarily be at the expense of quality. DevOps is the next major step to bring operations and development closer together and respond more quickly to changes. It is also a larger step towards a cultural change in software development and an important building block on the path to the cloud. With DevOps, beyond continuous integration and delivery, organizations gain greater visibility and feedback across the entire software development process as all results and steps are consistently automated and versioned. First of all, it is very helpful to see how far down the DevOps road the organization has come. We will explain this with the help of three simple questions. They do not give a full status assessment, but they can provide a clue as to where the organization stands:

1. Do developers have real-time access to troubleshooting information?
2. Does the production environment use the tools and methods which are provided or in use by the development team?
3. Do the developers and infrastructure team members work together in a partnership culture?

If the answer to these questions is "no," there is still no DevOps culture in the organization. If the answer to one or two questions is "yes," you've started changing to a DevOps organization.

Adopting DevOps

When adopting the DevOps concept, there are a few things you should remember. DevOps is more about culture and procedures than it is about organization, as you can see from its description so far. In any organization, culture is a difficult thing to change, but as with the successful adoption of an agile mindset, to truly excel in a DevOps implementation, you need to focus on the soft skills. It can be difficult to break down the barriers between two historically divided organizational regions (i.e., development and operations), but if you do it, you will be rewarded with higher-quality apps and happier users (Fazliu, 2018). There are some tools and techniques that will help incorporate procedures in such a way that they become an integral part of your everyday work. Tools may be used to enforce best practices, for example, the exchange of troubleshooting data between development and operations. You can do this by incorporating more software tools to see how the system performs not only in your development environment, but also in quality assurance and production. There are software tools that detect bugs, record timeouts, test device parameters, and so on during the code execution. The saved logs can be viewed using monitoring tools during operation. This helps you easily catch any code issues, which in turn enables you to repair bugs faster. The feedback loop is also smoother and you can be more flexible and receptive to market demands and modifications.

The Business Side Involvement

If the gap between development and operations has historically been large, the gap between the business side and development has been even wider (Rossberg & Olausson, 2014). For some reason, it is not easy for these two sections of a company to agree on a common approach when it comes to the requirements which have to be defined for applications and projects. The effect of such disagreements is that the project progress over time can be disastrous, and many stakeholders and end users have ended up feeling like they have not received the system they were looking for. Of course, complaining about a situation is easy, but it is important that we try to find a solution for it. One of the most effective means of change for decreasing the gap between business and development is to use agile processes. Although DevOps has no directly applicable solution to fill this gap, the fact that DevOps is in itself an agile approach helps the business and development sides come closer together.

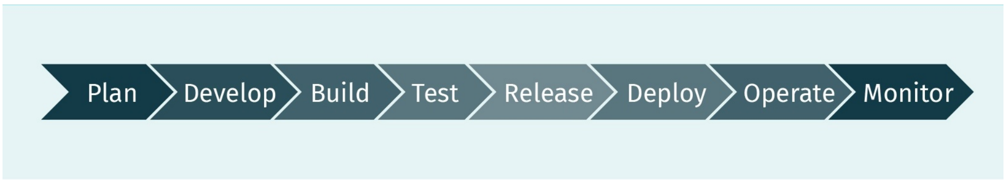
Communication

When building a cooperative DevOps environment, the problems for a company usually start with different ways of thinking in the departments, so effective communication would be very helpful in solving such problems. For this very reason, communication between development and operations must also be improved. On a cultural level, it is based on mutual respect and, on a communication level, on shorter distances from each other (the most effective communication is face-to-face, like two people sharing a whiteboard). Of course, IT operations must be able to formulate operational requirements. However, in order for these requirements to be met with understanding by the developers, having a respectful exchange about them is useful. Unsurprisingly, one of the basic requirements is to have a common language.

DevOps Pipeline

One of the most important prerequisites of creating a common language for better communication is the existence of a base concept which is accepted and understood by all stakeholders, especially development, operations, and quality assurance teams. This base concept can be defined as the automation of the transition from programming to operation, which has been expanded in many steps previously. This path from programming to operation is an important basis for DevOps and it is also referred to as the value stream or DevOps pipeline. The DevOps pipeline is the connection between developers and operations; in other words, it can bring development and operations closer together (Thedev, 2019). The following figure illustrates the DevOps pipeline.

Figure 14: DevOps Pipeline



Source: Kobdani (2020).

Each step of the DevOps pipeline is concisely described as follows:

Plan

Before the developers start coding, the beginning stage requires a thorough plan of the whole workflow. Product managers and project managers play an important role in this process. It is their responsibility to generate a production plan that will lead the whole team through the process. After collecting input and relevant information from users and stakeholders, the job is divided into a list of tasks. By segmenting the project into smaller, manageable parts, teams can produce results more quickly, fix problems as they occur, and respond to unexpected changes more easily. In the DevOps setting, teams work on sprints—a shorter period of time (usually two weeks) during which individual team members work on their assigned tasks.

Develop

In this stage, developers are beginning to code. Depending on the programming languages, developers install the needed IDEs (integrated development environment), editors, and other tools on their local machines to achieve full efficiency. In certain cases, developers need to adopt the coding styles and guidelines to maintain a clear coding pattern. This makes it easier for any member of the team to read and understand the code. Once developers are ready to push their code into the shared source code repository, they send a pull request. Team members will then manually check the newly submitted code and merge it with the master version.

Build

The build phase is important because it helps developers to identify code errors before they go through the pipeline and cause a bigger problem. In a typical case, the pull request initiates an automated process that compiles the code into a build, a deployable package, or an executable. Bear in mind that certain programming languages do not need to be compiled. For example, applications written in Java need to be compiled to run, while applications written in Python do not. If there is a code problem, the build will fail and the developer will be informed of the issue. If this occurs, the original pull request will also fail. Developers replicate this step every time they send to the shared repository to ensure that only the error-free code continues down the pipeline.

Test

If the build is successful, it moves to the stage of testing. There, developers run manual and automated tests to further verify the quality of the code. In most cases, the customer **acceptance test** is carried out; people communicate with the app as an end-user to decide whether the code needs additional changes before it is submitted to production. At the same time, automated tests could run security scanning against the application, search for infrastructure modifications and compliance with best practice hardening, test the application's performance, or run load testing. It is up to the company to decide what is important to the application and whether to carry out the tests at this stage, but this stage can be considered a test bed that allows you to plug in new tests without interrupting developer flow or affecting the production environment.

Acceptance testing
These are tests conducted to determine whether the requirements of a specification or contract are met.

Release

In a DevOps pipeline, the release stage is a milestone, the moment at which a build is ready to be deployed in the production environment. By this stage, a series of manual and automated tests have passed any code change, and the operations team can be assured that breaking problems are unlikely. They can choose to automatically deploy any build that reaches this stage of the pipeline, depending on the DevOps maturity of an organization. In order to turn off a new functionality, developers can use feature flags so that clients cannot see them until they are ready for action. Alternatively, as builds are released into production, a company may want to maintain control over them. They may choose to have a daily release schedule or, until a milestone is reached, just release new features. At the release stage, you can introduce a manual approval mechanism that only enables those individuals within an organization to approve a release into development. The tooling lets you configure this, how you want to go about things is up to you.

Deploy

The program is ready to be moved to production when the compilation enters the deployment point. If the code only requires minor modifications, an automated deployment method is used. If the framework has undergone a major redesign, however, the build is first deployed to a production-like setting to monitor how the recently added code will work. When releasing critical updates, it is often popular to adopt a blue-green deployment strategy. A **blue-green deployment** means having two similar development envi-

Blue-green deployment
This is a method for installing changes to a web, app, or database server by alternating production and staging servers.

ronments where the latest application is hosted by one environment while the modified version is hosted by the other. Developers should easily forward all requests to the relevant servers to release the modifications to the end-user. Developers may also easily return to the previous development environment without creating service disruptions if there are issues.

Operate

Now the latest version is live and is being used by clients. The operations team is now working hard, ensuring that everything runs smoothly. The environment automatically scales according to the number of active users, depending on the hosting servers and services. The company has also provided the clients with the possibility to use the services, as well as with tools to help collect their feedback to shape the product's future growth. No one knows what they want better than the clients, and customers make up the best testing team in the world, giving the application far more hours to test than the DevOps pipeline will ever do.

Monitor

The DevOps pipeline final stage is to monitor the running system. This builds on the customer feedback generated by data collection and analytics. We can also take some time for introspection and monitor the DevOps pipeline itself, possibly monitoring bottlenecks in the pipeline that cause confusion or affect the development and operations team productivity. All of this data is then fed back to the product manager and the production team in order to close the process loop. It would be convenient to assume that this is where the loop begins again, but the fact is that it is a continuous phase. There is no beginning or end, only a product's continuous progression over its lifetime, which only stops when individuals no longer need it.

DevOps Best Practices

An organization needs to pursue proper implementation plans to make the most of DevOps. We have already discussed what DevOps is, why we need it, and which steps the DevOps pipeline consists of. Now we can take a look at some of the best practices for DevOps (Patel, 2020).

Configuration management

Configuration management is an essential component of the DevOps process. It is the automation of all infrastructure entities and systems (e.g., servers, databases and other storage systems, operating systems, networking, applications, and software) that are used to install, administer, and maintain. For example, if we have an application that uses a database, the information about how to connect to the database can be kept in a configuration management system and be served wherever needed. Configuration management has some benefits, such as simplifying the setup of new environments, reducing the possibility of production configuration, and saving a lot of time for software creation instead of wasting time and resources using the infrastructure as code practice described below to initiate new services from scratch.

Release management

Release management in DevOps is about planning, scheduling, and managing the process of production and distribution of applications. From the start to the end of the process, all developers and IT operations cooperate, allowing for fewer, shorter feedback cycles and quicker updates. DevOps teams share responsibility for the services they provide, own their code, and assume on-call duties. Incidents are detected and handled more quickly, both during the release period and after, with software developers and IT specialists active during the entire delivery lifecycle and on request.

Continuous integration (CI)

In modern application development, several developers work on different features of the same app. The simultaneous merging of all source code branches in one day (also known as “Merge Day”) can be a huge amount of work and time. The reason for this is that application changes made by developers working separately can conflict with each other when performed simultaneously. This problem can be aggravated if each developer defines his own local integrated development environment (IDE) instead of creating a common cloud-based IDE as a team. With continuous integration (CI), developers can merge their code changes into a common “branch” or “trunk” of the application much more frequently, sometimes even daily. Once a developer’s changes are merged, they are validated in automated app builds and different levels of automation testing (typically unit and integration testing). This ensures that functionality has not been compromised. All classes and functions up to the various modules of the app must be tested. If the automated test detects conflicts between current and new code, CI can help resolve them more quickly and frequently (Patel, 2020).

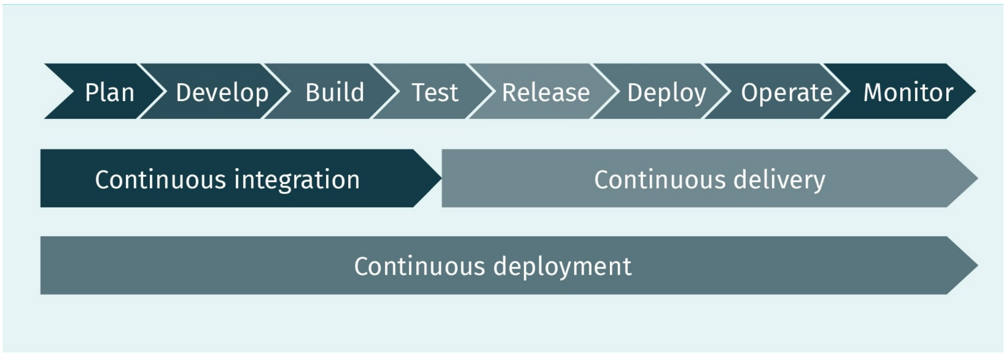
Continuous delivery

After automating builds and unit and integration testing for the continuous integration, continuous delivery can automatically release the validated code to a repository. Therefore, to ensure an efficient continuous delivery process, the CI must already be integrated into the development pipeline. The goal of continuous delivery is a code base that can be made available in a production environment at any time. With continuous delivery, every phase—from merging code changes to delivering production-ready builds—includes automated testing and code releases. At the end of this process, the operations team can quickly and easily deploy an app into production (Patel, 2020).

Continuous deployment (CD)

Continuous deployment can be seen as an extension of continuous delivery, in which production-ready builds are automatically released to a code repository. Continuous deployment also automates the release of an app into the production phase. Because the production phase in the pipeline is not preceded by a manual gate, automated testing in continuous deployment must always be well thought out. In practice, continuous deployment means that a developer’s app changes can go live within minutes of their creation, provided that they passed the automated test (Patel, 2020). This makes continuous integration of user feedback much easier. All interrelated CI/CD practices make an application implementation less risky because changes are released in parts rather than all at once. However, the upfront investment is considerable, since automatic tests must be written for the various test and release phases in the CI/CD pipeline. The DevOps CI/CD diagram shown below presents a simple comparison of continuous integration, delivery, and deployment.

Figure 15: DevOps CI/CD



Source: Kobdani (2020).

Infrastructure as code

Infrastructure as code is the activity of defining all software runtime environment and networking settings and parameters that can be stored and modified on request in simple textual format in the code repository (Null, n.d.). These text files are called manifests and are automatically provisioned and configured by infrastructure software to build servers, testing, staging, and development environments. Most significantly, all of these operations can be tracked via the code repository, which means the elimination of the decades-old issue of “works fine on my computer,” where the code that worked in testing does not work in production. Infrastructure as code guarantees continuity as all environments are automatically provisioned and configured with less chance for human error, which significantly speeds up software development and infrastructure operations and simplifies them.

Test automation

Automated testing of each codebase allows developers to run more tests, improves the pace of testing, and saves time spent on manual quality assurance. This process allows for early identification of bugs, bug-fixing, and increases the overall quality of apps. Several tools available for test automation—such as Selenium, Appium, or Junit—can also be combined with DevOps tools (Verona, 2016).

Continuous monitoring

Continuous monitoring suggests using multiple tools, dashboards, and notifications to track all systems and infrastructure, including real-time insights of various software-impacting metrics such as system performance, number of tests, success and failure rates, implementation status, error logs, as well as all graphical, tabular, and comprehensive report format information. Several tools such as Prometheus, Grafana, Nagios, Appdynamics, NewRelic, Splunk, and Logstash are available on the market to build a monitoring DevOps tool set (Elastic, n.d.).

Site Reliability Engineering

Site reliability engineering (SRE) and DevOps are two trending disciplines with quite a bit of overlap. Some have even named SRE a competing set of practices to DevOps in the past (Vargo & Fong-Jones, 2018). SRE incorporates the fields of software engineering and operations. They spend about half their time on development and half on operations. The SRE function allows for communication and sharing of information between development and operations, similar to the concept of DevOps but with unique additional goals (Wikipedia, 2020b).

2.3 Impact on Team and Development Structure

One of the main objectives of a DevOps organization is the smooth cooperation between the development and operations teams. In order for these teams to work together to develop, test, and deploy applications faster, DevOps was built to eliminate **silos** (DevOps, 2015). However, DevOps is much more than just a theory or a catchy abbreviation, as the system components go much deeper than that.

Silos
This is a management system that creates a structure in which teams concentrate on their own goals instead of working towards organizational objectives.

If we want to answer the question of what the ideal team structure for DevOps is, we have to consider a few things. The only way to ensure that the existing structure also works for DevOps is that Dev and Ops work together so that business goals can be met or exceeded. This, of course, looks differently for each individual business, which helps in the analysis of different models. You can find a better fit for the unique needs of the team by looking at the pros and cons of each model. When it comes to team structure, various factors play a role.

- Existing silos: Are there individually operating teams?
- Leadership: Who leads teams and what is their background in the industry? Do development and operations have the same priorities, or are they motivated by their leaders' individual experience?
- IT operations: Have activities been completely integrated with the company's priorities, or are they just seen as setting up servers and assisting the development team with their agenda?
- Knowledge gaps: Is the company equipped with the expertise and skills necessary to move towards a DevOps organization?
- Architecture: Do the architectural design principles fulfill the requirements needed to create DevOps teams?

Team Size

Amazon has attracted a lot of attention in the DevOps world by introducing the idea of the two-pizza team: The most powerful team is small enough that two pizzas could feed the whole team (Buchanan, 2019). The use of smaller team structures encourages the design of loosely coupled architectures and microservices that high-performance DevOps organizations prefer. However, reorganizing the organization's teams into smaller ones can be overwhelming, since changing team size is much easier said than done. One way to get started is to create a portfolio-based organization. The portfolio-based teams will consist of developers, operations, testers, conventional project managers, etc. Teams can be functional or role-aligned but should have the same approach aimed at improving the ability of the company to rapidly deliver value to its customers. This means that a company could have a vice president or technology director who designs a portfolio-based structure for the organization. Once the structure has been designed, the vice president manages a number of smaller, company-aligned DevOps teams headed by team managers.

Leadership

Suppose the instability in the company is very high and the improvement in communication needed to strengthen the DevOps structure is a significant one. The main challenge is to determine how the company can overcome the situation. The way leaders approach leadership can help to address the difficulties posed by the cultural change associated with DevOps implementation. Organizational changes are relatively hard to implement: A company-wide investment needs to take place and several departments may have to agree on a way to proceed. Change is not easy for any organization, but it is especially hard for those that do not interact well in the first place. Some of the largest failure predictors are

- change resistance,
- low change readiness, and
- weak employee commitment.

Transformational leadership is a leadership style in which leaders promote, empower, and motivate workers to make changes that help the organization grow and shape its potential success (Wikipedia, 2020a). This kind of leadership can be very successful depending on how team members respond to DevOps changes.

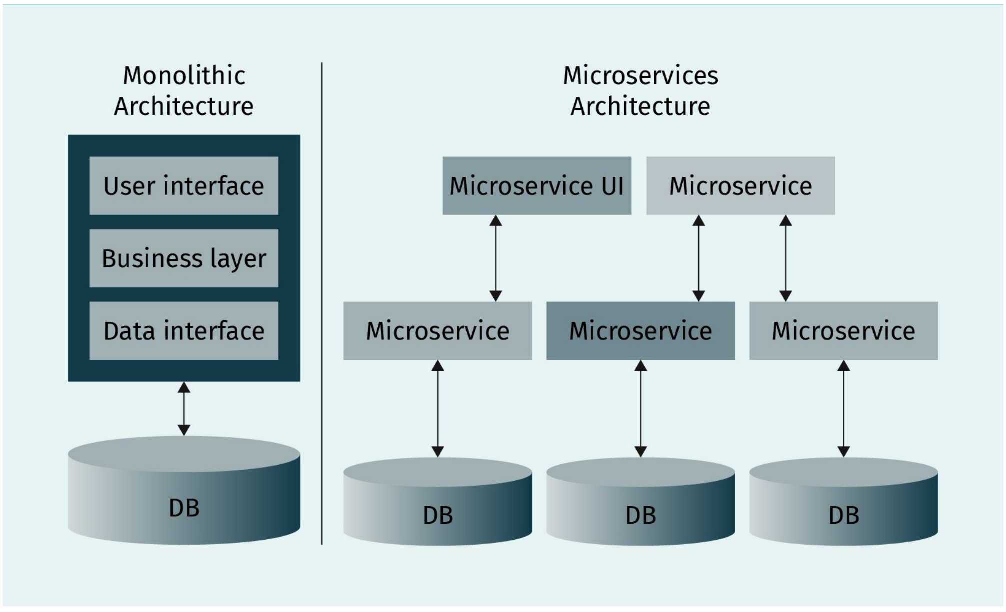
Right Mix

The first thing to do to create DevOps teams should be to determine skill gaps. This begins by determining which combination of roles and skills a company's employees would need to achieve the goals of the teams. In this sense, consider which members of the team occupy current positions, how many more people can be recruited, and which skills new employees need in order to fill the gaps. It is not only key technical skills or tasks that have to be filled which should be explored, but also the kinds of additional interpersonal skills and personalities which the leader thinks could help the workforce be more diverse.

Microservices Architecture

There are a number of ways to develop apps. For example, instead of implementing the project as a whole (i.e., monolithic application), it may make sense to distribute the task and put together small packages (microservices). Not only does the concept have a lot to do with the development of applications, it also plays a crucial role in the planning of agile project management and operations. However, the distributed architecture of microservices means that developers must also deal with the execution of applications in order to prevent errors in the linkage and synchronization of individual services. This is where DevOps can be really helpful. Both microservices and DevOps are two important trends that have emerged in the company; both are activities intended to provide the organization with greater resilience and operational performance. It would be fair enough to say that DevOps excellence is a key component of microservice excellence. The idea of microservices has emerged from a collection of popular DevOps ideas generated by a number of successful companies. In many cases, these companies began with monolithic applications, which quickly developed into decomposed services and communicated with other services through the RESTful APIs and other communication protocols. The following diagram compares the monolithic and microservices architecture (Anji, 2020).

Figure 16: Monolithic vs. Microservices Architecture



Source: Kobdani (2020).

In fact, IT departments may use microservice architectures to better respond to the requirements of business units and to deploy applications in a shorter timeframe. Close integration between production and IT operations, i.e., the implementation of the DevOps model, is very important here. A further benefit of the microservice approach is that smaller teams can also build and validate individual services and produce releases quickly, which also simplifies the implementation of DevOps. Microservices and DevOps complement each other in this way. In combination with the containers, the duo facilitates the deployment of a more scalable and productive infrastructure, provides applications that allow optimum use of infrastructure, and develops processes for designing and implementing these applications rapidly and at high quality.

12-Factor App

The 12-Factor app is a methodology or set of principles for the creation of scalable and resilient enterprise applications (Wiggins, 2017). It defines general principles and recommendations for the development of robust applications. The 12-factor app principles have become very common as they can be used for building microservices. The 12-Factor app is technology agnostic and fully compatible with microservices with a focus on DevOps.

The 12-Factor app follows the principles outlined below.

1. Codebase: Develop using a single codebase, completely tracked by a version control system; you should only have one repository for an individual application to make CI/CD pipelines simpler.
2. Dependencies: Declare and separate the dependencies explicitly; a configuration management system such as Chef (Chef, n.d.) or Ansible (Red Hat Ansible, n.d.-a) can be used to add system-level dependencies.
3. Config: Store configuration in the environment; in this way, building a CI/CD pipeline will be an easier and more flexible process.
4. **Backing services**: Treat backing services as attached resources. By following this principle, it is possible to swap the resource providers dynamically without impact on the system that increase the robustness of the CI/CD pipeline.
5. Build, release, and run: Strictly separate build and run phases; you can use CI/CD tools to automate the builds and deployment process.
6. Processes: Run the app as one or more stateless processes. By following these, the app will be highly scalable without any impact on the system. This decreases the possible conflicts that may happen between Dev and Ops when scaling is needed.
7. Port binding: Export services by **port** binding. This is more about creating a standalone application rather than deploying it to some external application server. The less dependent the CI/CD pipeline is on an external application server, the more robust it is.
8. Concurrency: Scale out using a **process model**; this is to encourage horizontal instead of vertical scaling.
9. Disposability: Maximize robustness with fast startup and **graceful shutdown**. By increasing the robustness of the system, you also increase the robustness of the CI/CD pipeline.
10. Dev/prod parity: Keep development, staging, and production as similar as possible. This reduces the risks of creating bugs in a specific environment that decrease the inconsistencies and conflicts may occur in the stages between development and production.

Backing service
Any service that the app consumes over the network as part of its operational activities is called a backing service.

Port
In computer networking, a port is a communication endpoint. It enables applications to communicate with each other over the Internet.

Process model
This model is a sequence of phases and tasks that constitute the entire life of the system.

Graceful shutdown
This is the ability of an operating system to perform the task of safely shutting down processes and closing connections when the running system is switched off by a signal.

11. Logs: Treat logs as streams of events. Logs are the key to troubleshooting in the production environment because they provide insight into the actions of the running program and are a communication tool between Dev and Ops.
12. Admin process: Run admin activities as inclusive processes. This increases the modularity of needed processes, which increase the modularity of the CI/CD pipeline components.

2.4 Building a DevOps Infrastructure

This section discusses how we can build a DevOps infrastructure. Although it would not be possible to discuss it comprehensively in just one section, this should at least give you some ideas to start with. As discussed before, DevOps is more of a culture that aims to promote cooperation and communication between the development and operations teams. The introduction of a CI/CD pipeline is often recommended for the technical implementation of the DevOps approach. The different steps in a CI/CD pipeline represent different subgroups of tasks which are divided into pipeline phases. These steps are generally similar to what we have in the DevOps pipeline, but depending on the technologies we select and use, they can differ from that. With an increasing number of CI/CD tools available on the market, it can be difficult for teams to decide what the right tools are. However, there are a few which have now been on the market for many years and that are used much more than others. If we want to select a specific one to discuss here, Jenkins would probably be the best choice.

CI/CD with Jenkins

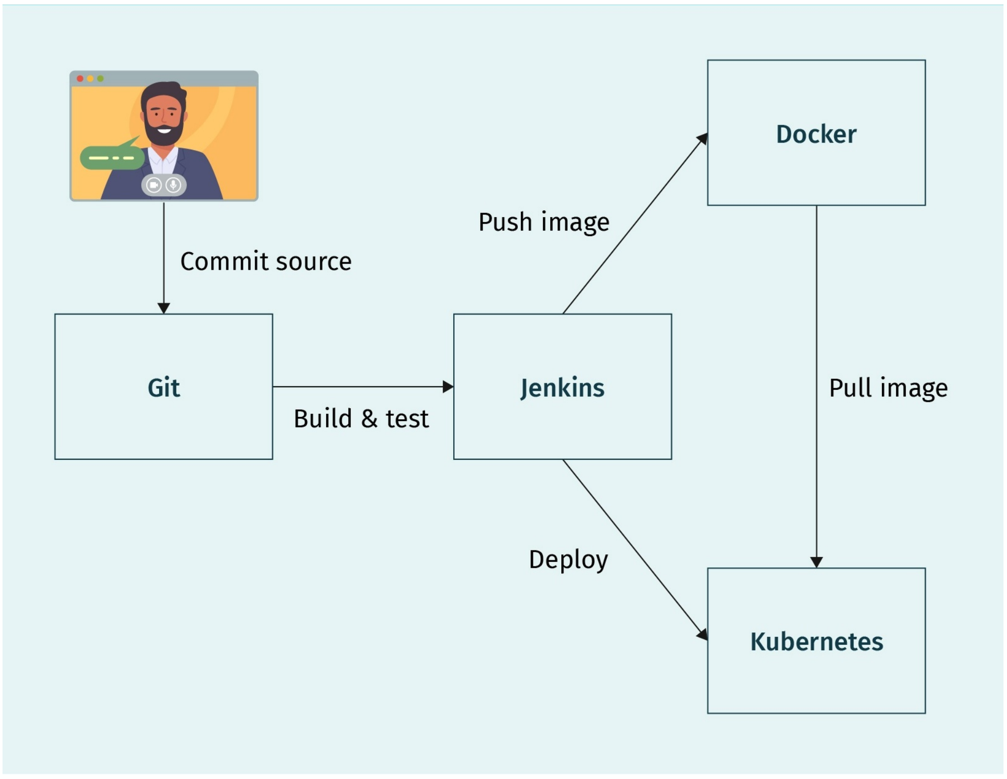
Jenkins is a leading open source automation server which provides hundreds of plugins to support the build, deployment, and automation of any project. It is written in Java and it runs in a web **container**; it also supports a wide range of different version control systems and has plugins for different technologies and deployment scenarios (Jenkins, n.d.-a). One of the most well-known combinations for building a robust infrastructure containing a CI/CD toolchain is Git, Jenkins, Docker, and Kubernetes.

Container
A container is a standard unit of software that packages code and all its dependencies so the application can run quickly and reliably from one computing environment to another.

Version control system
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Git is a free open-source **version control system**. It keeps track of projects and files as they change over time with the aid of various contributors (Git, n.d.). Docker is used to isolate applications using container virtualization. Containers are ideal for the independent deployment and execution of your microservice apps because they allow multiple parts of an app to run independently in microservices on the same hardware. At the same time, the individual components and lifecycles can be controlled much better (Docker, n.d.-a). Docker simplifies the deployment of applications because containers containing all necessary packages can be easily transported and installed as files. Kubernetes—also called k8s—is an open source platform that allows you to automate and orchestrate the operation of Linux containers (Kubernetes, n.d.). The following diagram shows the setup which is possible with Git, Jenkins, Docker, and Kubernetes.

Figure 17: CI/CD with Git, Jenkins, Docker, and Kubernetes



Source: Kobdani (2020).

Using the CI/CD setup shown above, the following automated steps are performed:

1. Code changes are committed to Git (Git, n.d.) or any other compatible version control system.
2. Each code change committed to GitHub automatically triggers a build in Jenkins. For example, Jenkins uses Maven (Apache Maven Project, n.d.; Jenkins, n.d.-b) to compile the Java code of your project, runs unit tests, and performs additional checks like code coverage, code quality, etc.
3. Once the code has been compiled successfully and all checks have been carried out, Jenkins creates a new Docker image and inserts (push) it to the Docker registry.
4. Jenkins will notify Kubernetes of the new image available for deployment.
5. Kubernetes takes (pull) the new Docker image from the Docker registry and deploys it.

To have a better understanding of what happens when using the above setting, we should take a closer look at Docker technology.

Docker

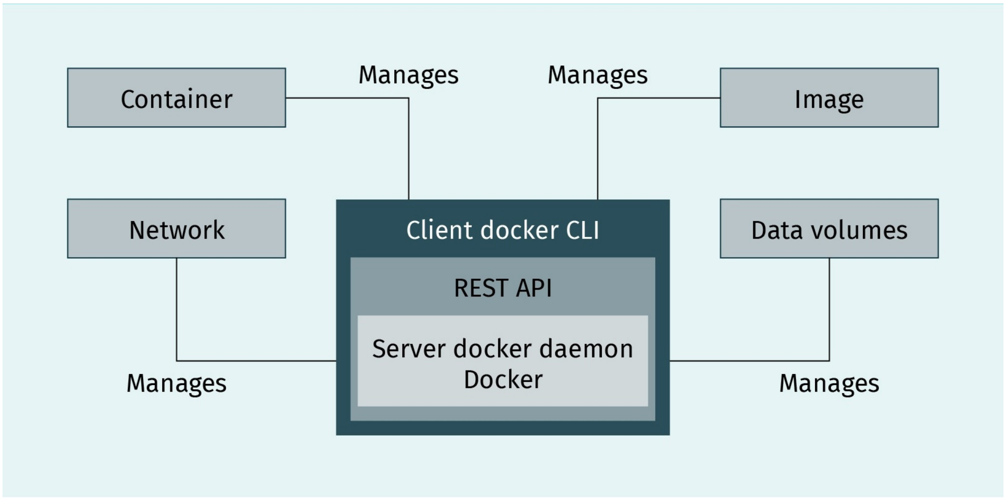
With Docker, we can package our application and the associated dependencies (runtime frameworks, libraries, etc.) into an image and run it on any machine. In simple terms, an image includes the elements needed to run an application. The basic requirement is that a Docker engine is installed on that machine. Docker will read the image and construct a container that runs on all platforms, whether on a phone, laptop, or cloud. The reasons for Docker's popularity are numerous (Tozzi, 2017). One reason is that Docker extends existing Linux container functionalities (for example, Docker provides versioning of images and containers). Additionally, Docker images can be quite easy to describe, build, and transport between systems and they are quite useful to manage very sensitive resources compared to its alternative solution, virtual machines. Virtual machines and containers vary in many ways, but the key difference is that containers offer a way to virtualize an operating system such that several workloads can run on a single instance of the operating system. With virtual machines, several operating system instances are run on virtualized hardware. Docker is open source, available for Windows, Linux, and Mac OS, and has a very wide community of tutorials and excellent documentation.

These are basic Docker-specific terms you should know (Docker, n.d.-b).

- Dockerfile: A text file that contains commands to create an image
- Image: A Docker image contains elements such as code, config files, environment variables, libraries, and run time that are required to run an application as a container.
- Container: A standardized unit that can be built on the fly to deploy a particular application or environment
- Docker registry: A service that contains Docker images and repositories
- Docker daemon: A server which is a type of long-running program called daemon process. A daemon (also known as background process) is a Linux or UNIX program running in the background.
- Docker REST API: Interfaces that programs can use to speak to the daemon and give it instructions
- Docker client: It provides a command line interface (CLI) that lets you create, run, and stop the containers on the Docker engine. The primary objective of the Docker client is to provide a way to pull images from the registry and to have them run on the Docker host.
- Docker engine: A client-server application comprised of the Docker daemon, the Docker REST API, and the Docker client. It forms the interface between the resources of the host and the running containers; Docker containers can run on any system on which the Docker engine is installed.
- Docker network: Docker includes support for networking containers through the use of network drivers.
- Docker volume: The preferred mechanism for persisting data generated by and used by Docker containers.

The following diagram illustrates the main components of the Docker system.

Figure 18: Docker Components



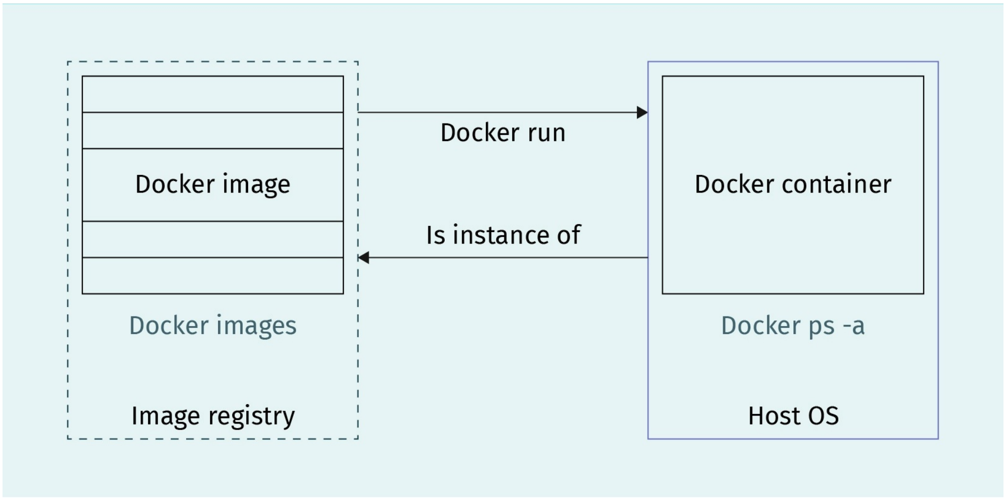
Source: Kobdani (2020).

The container is launched by running an image (containers are made out of images). An image is an executable package that includes all you need to run an application, namely

- the code,
- a runtime,
- libraries,
- environment variables, and
- configuration files.

The following diagram illustrates the concept of Docker images and containers and how they are related to each other.

Figure 19: Docker Image vs. Container



Source: Kobdani (2020).

The following Dockerfile is a simple example for building an image using a Python script.

Code
```
FROM python:3
ADD my-script.py /
CMD [ "python", "./my-script.py" ]
```

In the Dockerfile above:

- `FROM` indicates the image you need as the base of your Python script image. The argument `python:3` indicates the name and tag of the base image (the tag is specified after "`:`"). Here, the name of the base image is `python` and its tag is `3`.
- `ADD` adds the script to the image.
- `CMD` tells Docker to execute a command when the image is loaded.

To build an image based on the Dockerfile above (notice that the name "Dockerfile" is required), we may use the following command in the directory where the Dockerfile exists; that is what the "`.`" at the end of the command does (notice that we also need to place the application `my-script.py` in the current directory):

Code
```
$ docker build -t my-image .
```

After building the image, we may now run a container with the following command:

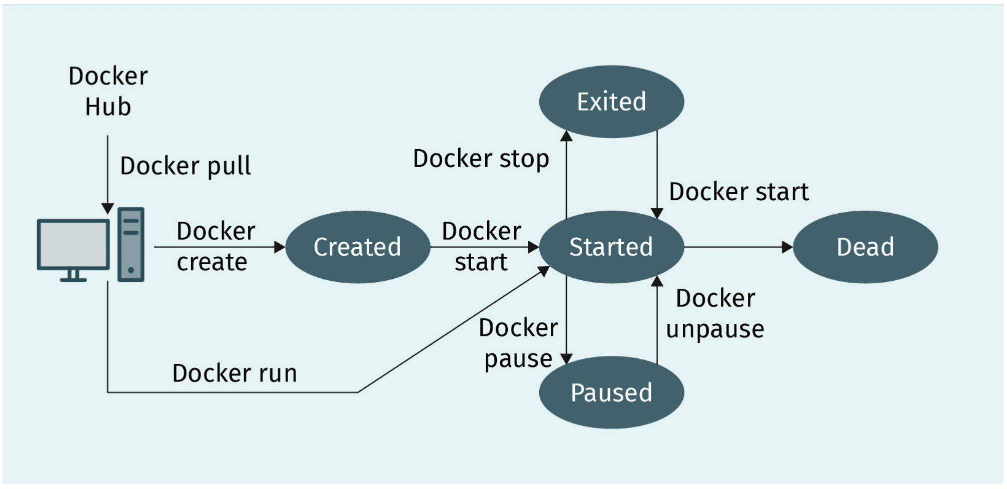
Code
```
$ docker run my-image
```

In the command above, `my-image` is the name of the image which we built before and which still exists in the local Docker repository. One interesting option of the `docker run` command is the `-it` flag (combination of `-i` and `-t` flags), which can be used to go inside a container. To enter a container by attaching a new shell process to it, use the following:

Code
```
$ docker run -it my-image bash
```

The diagram below illustrates a Docker container's lifecycle and the related commands.

Figure 20: Docker Lifecycle Overview



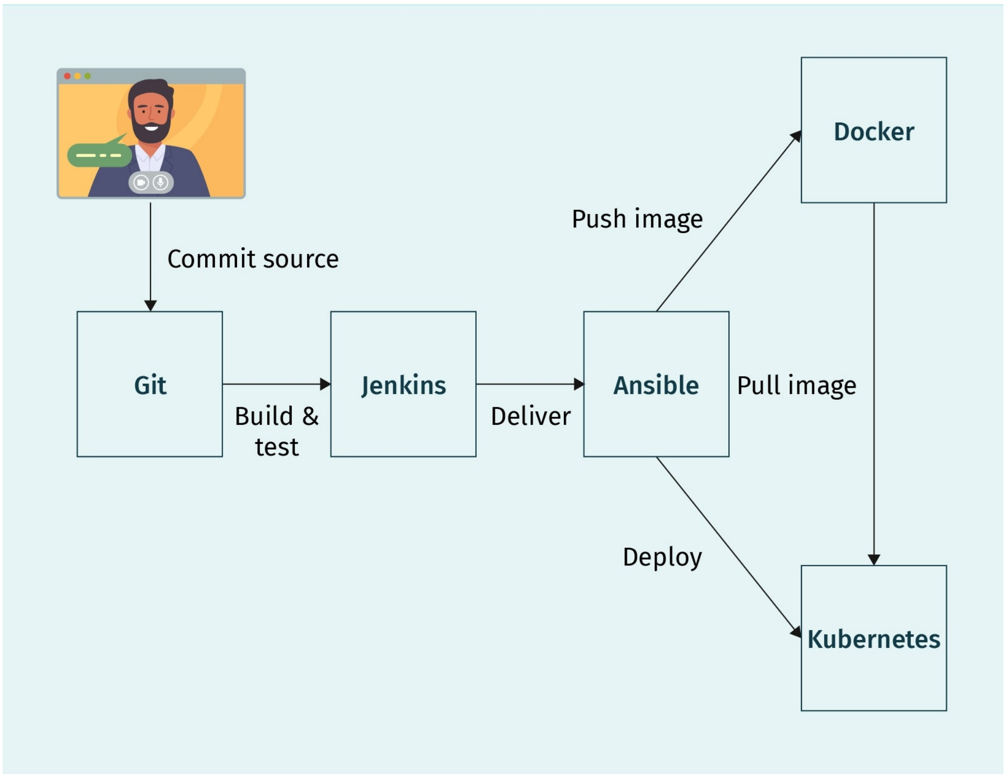
Source: Kobdani (2020).

Docker Hub is a location where the Docker images can be stored in order to be publicly accessed and used by developers to quickly produce fresh and composite applications. Thanks to the push and pull commands, we can write and read the images to and from the Docker Hub (Docker, n.d.-c).

Ansible

Although in the setup mentioned above Jenkins can be used directly to trigger a deployment, it is not the best solution. Since Jenkins is more of a continuous integration tool, using it for configuration management and continuous deployment presents some limitations. An alternative solution would be to use another tool besides Jenkins for configuration management and continuous deployment, for example Ansible or Chef. Ansible is one of the methods that are preferably used to automate operations that would otherwise have to be done manually with a considerable time investment and without adequate quality control (Red Hat Ansible, n.d.-b). The following diagram shows a CI/CD setup that uses Ansible for continuous deployment.

Figure 21: CI/CD with Git, Jenkins, Ansible, Docker, and Kubernetes



Source: Kobdani (2020).

Ansible is a software for the central management and administration of distributed servers. The community version of Ansible itself is license-free as OpenSource software within the Linux administration. In addition to the community edition of Ansible, there are other editions available from the manufacturer (e.g., Redhat), which are subject to license and provide, for example, a dashboard or workflows. Ansible—along with Puppet and Chef—is one of the most well-known software products with which distributed systems can be administered. Compared to Puppet and Chef, however, Ansible presents several advantages (Arora, 2020):

- Ansible does not need a central component; one computer is sufficient to access the servers that have to be managed via SSH (or secure shell, a network protocol for encrypted connections).
- The training effort for Ansible is significantly lower than for Chef or Puppet.
- For Ansible there are many ready scripts (playbooks), which can be downloaded mostly for free (for example, from GitHub).

In order for Ansible to function properly, we need the following components:

- Workstation/server: When using Ansible for daily work, it is recommended to install it on a computer or server where Linux is installed. This can be the workstation of the Linux administrator or another computer from which the servers that have to be managed can be easily reached.

- Network: In order for Ansible to access the servers to be managed from the administration installation, they must be accessible via a network. It does not matter whether the devices can be reached via the Internet, the LAN, or a VPN.
- SSH keys: The communication between Ansible and the remote hosts is mainly done via SSH. To allow Ansible on the central host to access the remote servers without a password, there must be an SSH connection using a certificate (how to set this up is explained below).

Ansible playbooks generally use Yaml and are particularly well-suited for the deployment of complex applications because they allow for the easy management of configuration and provide a multi-machine deployment framework that can be repeated and used multiple times. For example, the following Playbook describes the installation of NGINX:

Code
```
- hosts: local
  tasks:
  - name: Install Nginx
    apt: pkg=nginx state=installed update_cache=true
```

Terraform

Ansible, Puppet, or Chef are configuration management tools built to install and manage software on existing servers. They cannot be used for provisioning, where the building, changing, and versioning of infrastructure is the main focus. Terraform is a well-known provisioning software, meaning it is built to provision the servers themselves, as well as the rest of the infrastructure, such as load balancers, databases, etc. (Terraform, n.d.). The emphasis on configuration management or provisioning means that for some types of tasks, this software would be a better match (Brikman, 2016). More and more companies today are using cloud solutions to implement work environments or even complete IT structures (Rimol, 2019). Infrastructure as a service is in many cases the simplest and least expensive way to create the basis for planned projects. In addition, cloud solutions make it possible to react quickly to current requirements. While the underlying components (such as servers, firewalls, or load balancers) in the provider’s data center are static, they can be changed dynamically in the virtualized cloud environment, thereby giving customers the option of increasing or reducing resources as needed at any time. To ensure this flexibility, the providers provide APIs that allow the leased infrastructure environment to be scaled at any time with the appropriate software—freedoms that are attractive but also involve a high level of administrative effort. Terraform is the ideal solution to minimize this effort in the long term.

2.5 Building Scalable Environments

Scalability is one of the key objectives in a modern infrastructure because an organization that can scale consistently is one with a great potential for growth. Scalability means that a company can configure its systems to grow during high demand and to scale down when demand falls. Some activities typical of DevOps make this approach the right choice in order to achieve the optimal degree of scalability. These activities include communication opportunities, faster performance orientation, increased opportunities for creativity, and a faster release of apps. When you are operating at scale, container orchestration is an essential requirement. Container orchestration automates the provisioning, management, scaling, and networking of containers. Companies that need to deploy and manage hundreds or thousands of containers and hosts can benefit particularly from container orchestration (Red Hat Ansible, n.d.-c). You can use container orchestration in any environment where you use containers. Using container orchestration, you can deploy an application in different environments without special customization. With microservices in containers, you can orchestrate your services, such as storage, networking, and security, more easily (VMware, n.d.).

Container orchestration allows you to automate and manage the following tasks:

- Provisioning and deployment
- Configuration and planning
- Resource allocation
- Container availability
- Scaling or removing containers to evenly distribute workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring of the container status
- Configuring applications based on the container in which they will run
- Securing interactions between containers

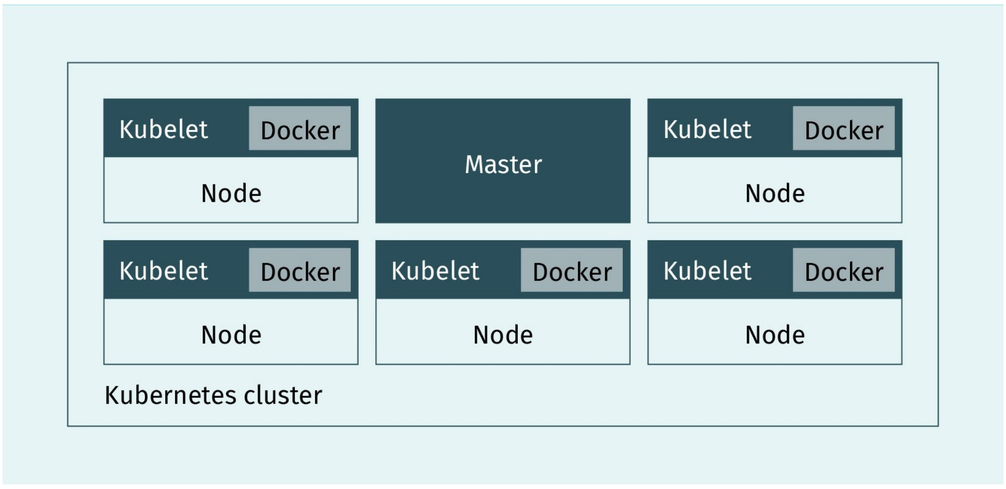
Container orchestration tools provide a framework for managing containers and microservice architectures on a large scale. There are many container orchestration tools that can be used for container lifecycle management. Some popular options are Kubernetes, Docker Swarm, and Apache Mesos (Apache Mesos, n.d.).

Kubernetes

Kubernetes is an open source tool for the orchestration of containers, originally developed and designed by Google. With a Kubernetes orchestration, you can develop application services that span multiple containers, plan and scale containers across clusters, and monitor their health over time (Kubernetes, n.d.). Kubernetes eliminates many of the manual processes associated with the deployment and scaling of containerized applications. You can have cluster groups of hosts (either physical or virtual machines) running containers because Kubernetes provides the right platform to manage these clusters easily and efficiently. More generally, with Kubernetes you can implement an infrastructure in your production environments that is completely container-based and reliable. Kubernetes clusters can include hosts in public, private, or hybrid clouds. For this purpose, Kubernetes is the perfect framework for hosting **cloud-based applications** that require rapid scaling. Kubernetes also facilitates the portability of workloads and load balancing by allowing applications to transfer without re-developing them. The following diagram illustrates a Kubernetes cluster:

Cloud native
An approach that ensures that applications are designed and developed for the cloud computing architecture.

Figure 22: Kubernetes Cluster



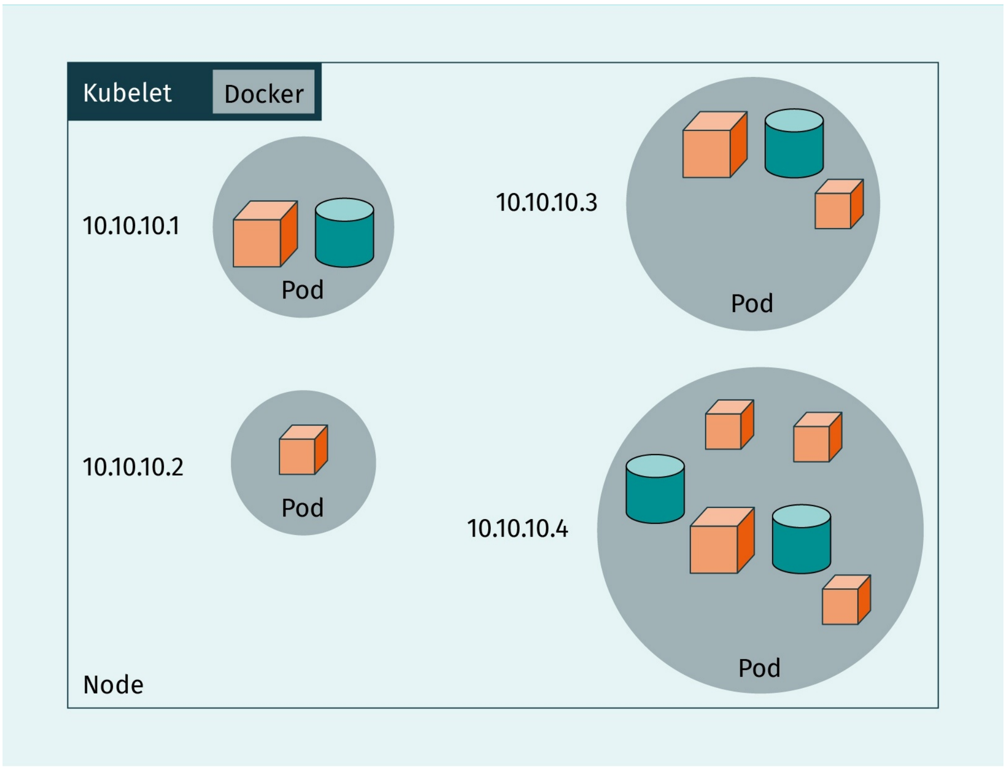
Source: Kobdani (2020).

A Kubernetes cluster consists of two types of resources:

- The master, i.e., the coordinator of the cluster
- Nodes, i.e., the workers that run applications

The master is responsible for the administration of the cluster. The master coordinates all activities in your cluster, such as scheduling applications, managing the desired status of applications, scaling applications, and rolling out new updates. A node is a VM or a physical computer that serves as a working machine in a Kubernetes cluster. Each node has a "kubelet," an agent to manage the node and communicate with the Kubernetes master. The node should also have tools to handle container operations, such as Docker. A Kubernetes cluster that handles production traffic should consist of at least three nodes. If you deploy applications on Kubernetes, instruct the master to start the application containers. The master plans to run the containers on the nodes of the cluster. The nodes communicate with the master via the Kubernetes API that the master provides. End users can also use the Kubernetes API to interact with the cluster directly. Pods are the smallest computing units you can build and manage in Kubernetes. A pod (as in a pea pod or a pod of whales) is a group of one or more containers, with shared storage and network resources, and a specification on how to operate the containers (Kubernetes, 2020). The contents of the pod are often co-located and co-scheduled. A pod models an application-specific "logic host," which includes one or more application containers that are relatively tightly coupled. In non-cloud environments, programs running on the same physical or virtual machine are similar to cloud applications running on the same logical host. A pod always runs on a "node." The node may have several pods, and the Kubernetes master automatically manages to schedule pods across the nodes in the cluster. The automated scheduling of the master is done by considering the resources required for each node. The following diagram illustrates how pods are running on nodes.

Figure 23: Kubernetes Pods



Source: Kobdani (2020).

Docker Swarm

The concept of the Docker Swarm is similar to Kubernetes in that they are both used within a cluster to deploy and manage containers. Both Kubernetes and Docker Swarm are designed to coordinate node clusters on a scale in a production environment. Swarm is a software developed by the Docker developers, which combines any number of Docker hosts into a cluster and enables centralized cluster management and container orchestration. Until Docker version 1.11, Swarm had to be implemented as a separate tool, while newer versions of the container platform support a native Swarm mode. The Cluster Manager is thus available to every Docker user with installation of the Docker engine (Docker, n.d.-d). Docker Swarm is based on a master-slave architecture. Each Docker cluster (the swarm) consists of at least one manager node and any number of worker nodes. While the Swarm manager is responsible for managing the cluster and delegating tasks, the Swarm workers take over the execution of work units (tasks). Container applications are distributed as services to any number of Docker accounts. In Docker Swarm terminology, the term “service” describes an abstract structure used to define tasks that are to be executed in the cluster. Each service consists of a set of individual tasks, each of which is processed in a separate container on one of the nodes in the cluster. When you create a service, you determine which container image it is based on and which commands run in the container. Docker Swarm supports two modes in which Swarm services are defined: replicated and global services.

Replicated services

A replicated service is a task that runs in a user-defined number of replicates. Each replica is an instance of the Docker container defined in the service. Replicated services can be scaled as users create additional replicas. For example, a web server, such as NGINX, can be scaled to 2, 4, or 100 instances with a single command line, as needed.

Global services

When a service is running in global mode, each available node in the cluster starts a task for the corresponding service. When a new node is added to the cluster, the Swarm manager immediately assigns a global service task to it. For example, global services are useful for monitoring or logging applications. A central application of Docker Swarm is load balancing. In Swarm mode, Docker has built-in load balancing capabilities. For example, if you run a NGINX web server with 4 instances, Docker will intelligently distribute incoming requests to the available web server instances.

Kubernetes vs. Docker Swarm

Kubernetes supports higher demands with more complexity, while Docker Swarm provides a simple solution that is easy if you are just getting started. Docker Swarm has become very popular with developers who prefer fast deployments and simplicity. At the same time, Kubernetes is used in production environments by numerous high-profile internet companies operating common services (Mangat, 2019). Both Kubernetes and Docker Swarm will run many of the same services, but some specifics can require slightly different approaches. By comparing them, you can make a decision in order to choose the right tool for your container orchestration.

SUMMARY

In this unit, we learned about DevOps, we saw the difficulties faced by development and operations teams in a traditional environment and how DevOps can help in such a scenario. The important aspects of the DevOps culture, including the DevOps pipeline, configuration management, release management, continuous integration and continuous delivery, infrastructure as code, test automation, and continuous monitoring have also been explored in this unit. We have covered the impact of DevOps on team and development structure by discussing team sizes, leadership, microservices architecture, and the 12-factor app principles. This unit also offered a very short introduction to how we can build a DevOps infrastructure and scale it up, as well as to two CI/CD pipelines containing Git, Jenkins, Ansible, Docker, and Kubernetes.

UNIT 3

SOFTWARE DEVELOPMENT

STUDY GOALS

On completion of this unit, you will have learned …

- different testing paradigms in the software development lifecycle.
- different approaches to testing while developing, such as test-driven development (TDD) and behavior-driven development (BDD).
- the terms continuous integration (CI), continuous delivery (CD), continuous testing (CTe), and continuous training (CT).
- how to apply the CI/CD approach to improve machine learning systems.
- how to track changes in a software development project using tools.
- what integrated development tools (IDEs) are and some example of development tools.

3. SOFTWARE DEVELOPMENT

Introduction

After finishing with the development of a machine learning (ML) solution, the project team does not deploy the software immediately to the production environment to serve customers. There is another step between the development and deployment: testing. The testing procedure is a cross-functional and continuous activity in which the whole team is involved. In an ideal case, the testing team will collaborate with the developers and users to develop and improve the automated testing at the very beginning phases of the software project. These automated tests will be developed by the testing team before development of the modules by the development team. If the developed modules pass the relevant tests, it demonstrates the successful implementation of the required functionalities in the software. The process of identifying and prioritizing the project's risks and designing measures to mitigate those risks is called testing strategy. A well-designed testing strategy ensures working software functionality (meaning fewer bugs and lower support costs), and defines a framework for development practices. Different testing types from the perspective of different stakeholders of a project will be discussed at the beginning of the unit, after which we will discuss how to test and develop in parallel. The scenario for a machine learning solution project as a non-deterministic system is more complex and requires a revision of the conventional testing methods. At this point, we will also learn how to automate the development pipeline. We will then discuss automated development, in which it is even more crucial to keep track of changes.

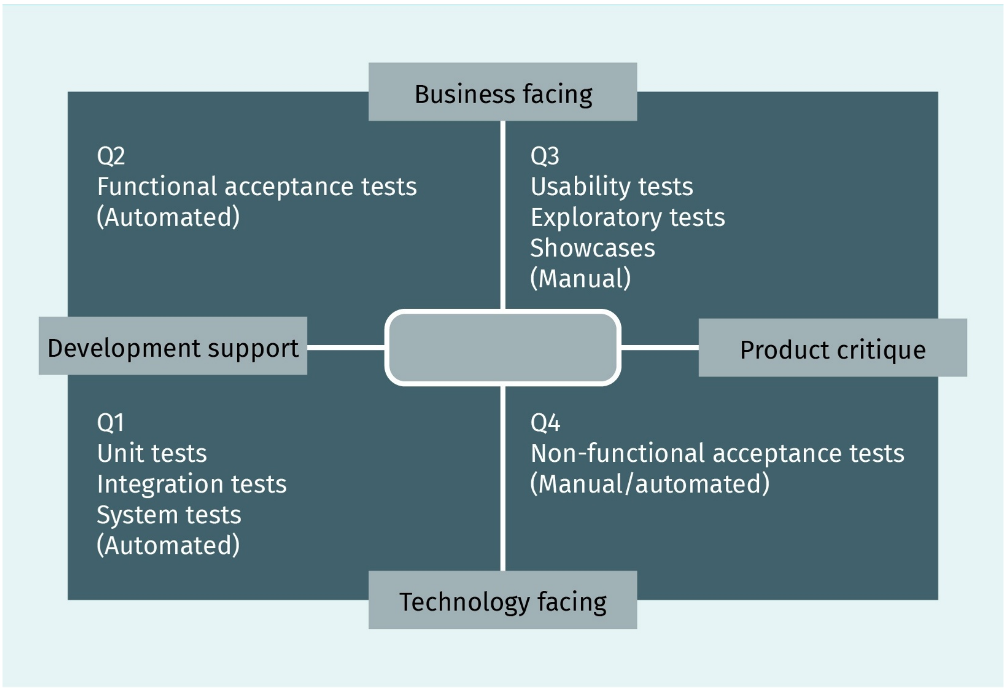
3.1 Testing Paradigms and Monitoring

To deliver a high-quality software solution, Brian Marick introduced several test scenarios shown in the figure below (known as Marick's quadrant), which the project team should implement and perform before releasing the software solution (Humble & Farley, 2015). These tests are categorized into two dimensions:

1. Tests that support development (help the developers to build the software with confidence) and tests that critique the product (help to discover insufficiencies and deficiencies in the software programs)
2. Tests that are business-facing (developed from the perspective of a business person) and tests that are development-facing (developed from the perspective of a developer)

In the following, we will discuss these four scenarios in more detail.

Figure 24: Marick’s Test Quadrant



Source: Alvarid (2020), based on Humble & Farley (2015).

Quadrant 1 (Q1)

Among the technology-facing tests that support the development team there are unit tests, integration tests, and system tests. These tests are normally written by the development team. In a unit test, a unit of the software will be tested in an isolated environment without interaction with the other components (units) of the system, i.e., to perform a unit test, interactions with other units of the system will be simulated (to provide the required inputs). This isolation results in the fast execution of unit testing. However, due to this isolation, unit testing cannot ensure the functionality of the unit as it interacts with other software components. Therefore, we need another suite of tests, namely integration tests. In an integration test, a wide range of functionalities embedded in different units of the software product will be tested. In such a test, not only the units’ functionalities but also the interactions and the interfaces between the units will be tested. Through integration testing, it is ensured that each independent unit of the software solution works seamlessly with the services it depends on. An integration test of the system under test (SUT) could be executed against the real external systems it depends on or against a **test harness** as a part of the software product. System testing is the testing of the software product as a whole to assess its compatibility with the client’s requirements specified in the software requirement specification (SRS) document. In integration testing, we perform a **functional test** on a collection of the units and their interaction; in system testing, we test the system as a whole. System testing follows integration testing. Before deploying a software solution into the production environment, a deployment test is crucial. Such a test will check if, for example, the software solution is installed and configured correctly and is able to communicate with the required services.

Test harness
In software quality assurance, this is a collection of software and test data that is used for automated testing of a software under different environmental conditions as well as monitoring its behavior and outputs.

Functional test
The requirements that specify what a software system must or must not do are called functional requirements. Non-functional requirements specify "how" the system should do things.

Quadrant 2 (Q2)

Tests that support the development team on the business-facing side are commonly known as functional acceptance tests. An acceptance test ensures the operation of some pre-defined criteria such as functionality, capacity, usability, and security. An acceptance test that concerns a functionality criterion is called a functional acceptance test. When such a test is passed, the development team and users can be sure that the functionality of the developed unit is implemented correctly. End-to-end testing (E2E testing), when the developers test if the whole software product from beginning to end behaves as expected, is also an example of this category (Q2). In writing this test, the system's dependencies are defined, and the test ensures that all pieces of the software solution work together according to the specifications. The main goal of E2E testing is to test the software product from the viewpoint of the user by simulating real user scenarios (Katalon, 2020). In system testing, only the specific software system is tested; in E2E testing, the specific software system is tested together with the connected external systems.

Quadrant 3 (Q3)

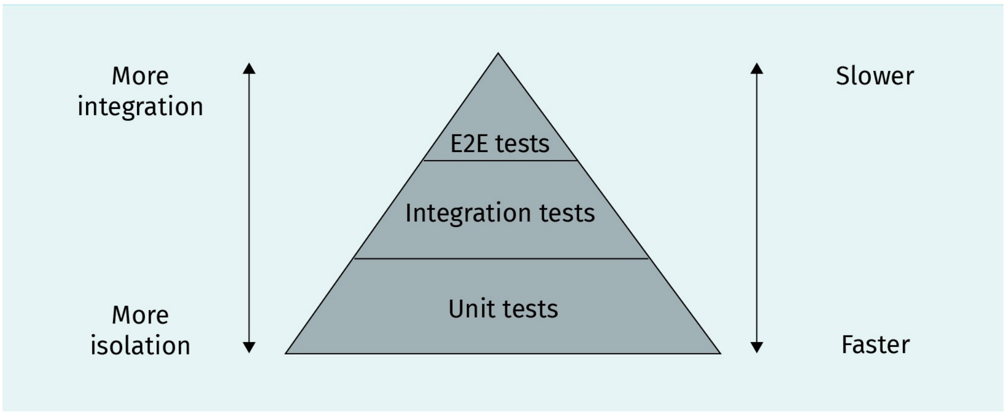
Business-facing tests that critique the project are mostly manual tests that verify if the software product delivers the expected values to the customer. It checks not only the specifications but also whether the specifications are defined correctly. Some examples of tests in this category are showcases, exploratory testing, and usability testing. At the end of each iteration, the development team performs showcases to demonstrate the new functionalities to users. This demonstration of functionality during the development phase prevents any misunderstanding or specification problem. Another example of testing, exploratory testing, is defined by J. Bach (2003) as manual testing where "the **test engineer** actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests" (p. 2). Exploratory testing results in new sets of automated testing. Finally, usability testing is performed to test how the users could easily accomplish the defined goals using the developed software.

Test engineer
In a software project, the person or team responsible for defining, executing, and reporting the test procedure during the software development lifecycle is called the test engineer (or team).

Quadrant 4 (Q4)

Technology-facing tests that critique the project verify the nonfunctional criteria of the software system like capacity, availability, and security. Nonfunctional acceptance testing is designed for this purpose. This type of testing could also be fully automated, but it is performed less frequently than functional acceptance testing and usually at the end of the development pipeline. After getting familiar with different types of software testing, now the question is how to design an automated testing suite. An effective automated testing strategy could be divided into three levels, structured by the so-called test automation pyramid as shown in the figure below (Cohn, 2009). In this model, the unit test forms the base of this test automation pyramid. The next level is integration testing, and at the top of the pyramid is the end-to-end testing (E2E). The testing pyramid is the original representation of this approach and it could be extended to contain more automated test methods (mostly from the Q1 and Q2 regions in Marick's test quadrant).

Figure 25: Test Automation Pyramid



Source: Alvarid (2020), based on Humble & Farley (2015).

Machine Learning Models Testing

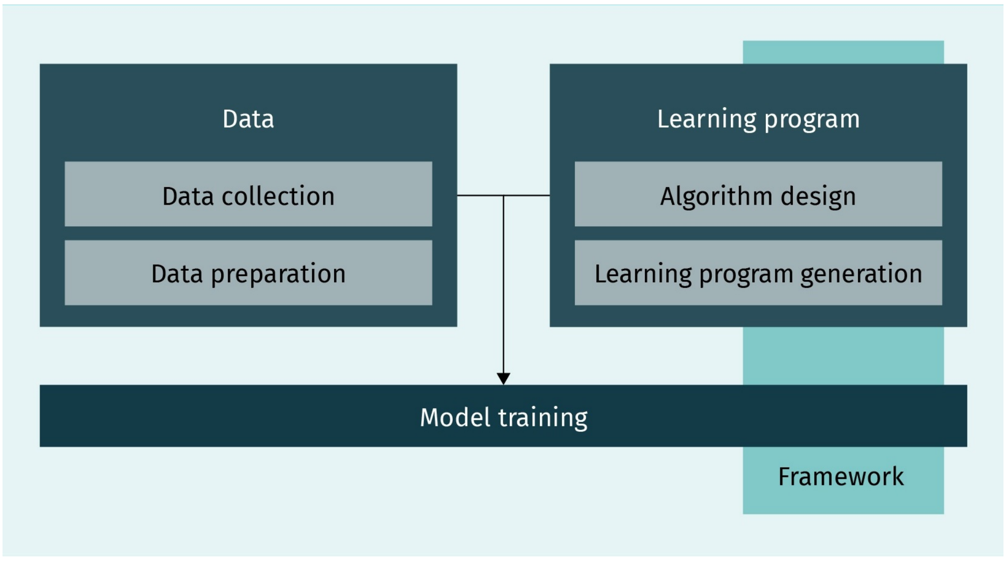
Unlike traditional software products, machine learning (ML) models are **non-deterministic**, but there are several possibilities for the transition to the next state for the same input. As in such a software solution, the responses of the system are adapted to what the system has learned in previous transactions. Therefore, in machine learning solutions, the tester should test data, code, and the learning program, as well as the frameworks (e.g., **TensorFlow**) that support the ML development. In addition, developing test oracles is time- and labor-consuming, as domain-specific knowledge is required (Zhang et al., 2020). An oracle is a mechanism used in software testing and software engineering to determine whether a test was successful (Kaner, 2004). It is used by comparing the output of a system under test, given as input to a specific test case, with the result that the product should provide (determined by the oracle).

Non-determinism
This is a concept from theoretical computer science in which algorithms or machines cannot only perform exactly one calculation for a certain input (deterministic). There are instead several possibilities for the transition to the next state for the same input.

TensorFlow
This is an end-to-end open-source platform for machine learning that includes a comprehensive, flexible ecosystem of tools, libraries, and community resources. It lets researchers push the state-of-the-art in ML and developers easily build and deploy ML -powered applications (Tensorflow, n.d.).

The components involved in a machine learning model building are represented in the figure below (Zhang et al., 2020). To test an ML software solution, testers should test all these components, as well as their interactions.

Figure 26: Components Involved in the ML Model Building



Source: Alvarid (2020), based on Zhang et al. (2020).

Testing data

In testing data components of an ML model, test engineers should check issues such as whether the data are sufficient for the training; if the data fairly represent future data; whether data contain a lot of noises (such as biased data); and if there is a skew between the training and test data (Zhang et al., 2020). Although it is mostly the task of the ML developer to design the right test cases, there are some general hints (Heck, 2020).

- Data type and schema: ML engineers always check the format, type, and schema of data to ensure data quality and integrity. This test helps the ML engineers to ensure that data are clean from malformatted inputs (Kim et al., 2018).
- Implicit constraints: ML engineers often examine certain constraints, like assertions in software testing. Such constraints are not regarding single data points, but rather how the subgroups of data relate to other subgroups (Kim et al., 2018). For example, the number of downloaded software licenses from a specific region should be equal to the number of purchased licenses.
- Testing platforms: There are some ML testing frameworks from companies like Google's TFX (Breck et al., 2019) and Apple Overton (Ré et al., 2019). For instance, Google provides a data validation system that lets developers define a data schema to check data properties and to generate synthetic data to test the model (Breck et al., 2019).

Testing the learning program

A learning program (ML model) is composed of the algorithm and the code to implement and to configure it. A trained ML model should be tested according to metrics such as correctness, fairness, and interpretability (Zhang et al., 2020). Correctness includes accuracy, precision, and recall. To explain the difference between precision and recall, consider an ML algorithm that classified 100 tumors as malignant (the positive class) or benign (the negative class): true positives (TPs), false positives (FPs), false negatives (FNs), and true negatives (TNs) (Google, 2020b). Precision is defined as the proportion of actually correct positive identifications (TP/(TP+FP)). A recall is the proportion of the actual positive cases that have been identified correctly by our algorithm (TP/(TP+FN)). A fair ML model makes fair predictions across various demographic groups (Dwork, 2012). IBM has developed a tool for ML fairness called AI Fairness 360 (IBM, n.d.-a). This tool assists the ML developer to examine, report, and mitigate discrimination and bias in ML models throughout the AI application lifecycle. Finally, an ML model is assumed as an interpretable model when an observer/user can understand the cause of the decision produced by it. There are some tools for checking interpretability such as the Machine Learning Interpretability module from H2O (H2O, n.d.).

ML Model Monitoring

The ML system should be monitored to detect any unexpected behavior due to changes in the input data. Monitoring is essential for models that automatically digest new data in a real-time fashion during the training, therefore, it is always required that these models also predict in real-time. Four classes of monitoring ML systems could be defined as follows (Gade, 2019):

1. Feature monitoring, to guarantee that model features are stable over time, data invariants are stable, to have a continuous overview of statistics
2. Model operations monitoring, to detect staleness, regressions in serving latency, throughput, RAM usage, etc.
3. Model performance monitoring, to detect regressions in prediction quality
4. Model bias monitoring, for detection of unknown bias

Amazon SageMaker Model Monitor is an example of an ML monitoring tool that allows developers to detect and remediate **concept drift** (Amazon, n.d.-b). SageMaker Model Monitor automatically detects concept drift in deployed ML models and provides detailed alerts that help identify the source of the problem.

Concept drift
If the data being used to generate predictions differs from data used to train the model, the patterns the model uses to make predictions no longer apply and this is called concept drift.

3.2 Approaches to Development and Testing

In software engineering, a software development methodology (also known as lifecycle) applies to the software development project and breaks it into separate phases or stages. Each phase includes activities with the purpose of more efficient planning and management of the project. The waterfall model is one of the traditional methodologies, while the **agile models**, like Kanban and Scrum, are the more modern approaches (Lumen, 2020). In this section, we briefly discuss three software development techniques that support agile development methodologies by enhancing the testing practice and assisting automated testing:

Agile model
In this approach, specifications, requirements, and code change dynamically through collaboration between cross-functional teams.

- Test-driven development (TDD)

- Behavior-driven development (BDD)
- Acceptance test-driven development (ATDD)

Test-Driven Development (TDD)

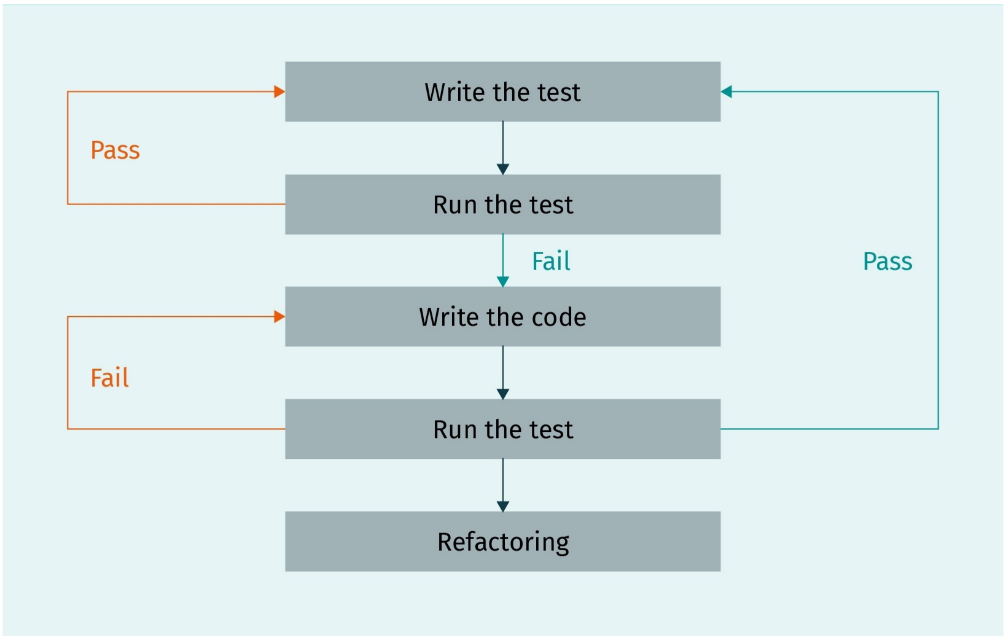
Test-driven development is a testing methodology performed from a developer's perspective. In this method, a quality assurance engineer begins designing and writing test cases for every functionality of the software product. This technique tries to answer a simple question—is the code valid? The main purpose of this technique is to revise or write fresh code only when the test fails, resulting in less duplication of test scripts. This technique is most common in agile development. In a TDD approach, automated test scripts are written before functional pieces of code. The TDD lifecycle can be split into the following steps (Beck, 2014):

1. Write the test. This test should define (or improve) some specific functionality. For this reason, the test developer should be familiar with the product's requirements. This is what differentiates TDD from traditional development approaches, i.e., first writing the test and then the code.
2. Run the test and verify that it fails, as the functionality has not been written yet.
3. Write the code for the functionality that should pass the written test. In this step, the developer should write the simplest code that will make the test work.
4. Test the new code with the written test. It should pass the test.
5. **Refactor** to remove duplicate code and clean up the code during the TDD process. In this step, all naming conventions should also be checked and corrected.

Refactoring
In the software development lifecycle, the process of cleaning and simplifying the design of an existing code, without changing its behavior and functionality, is called refactoring.

The aforementioned steps are illustrated in the following flowchart:

Figure 27: TDD Lifecycle

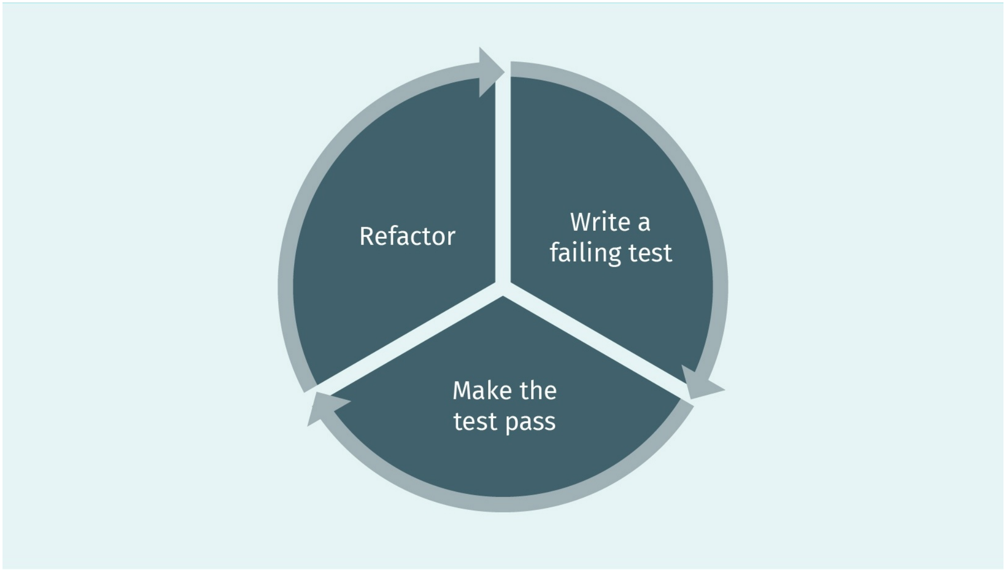


Source: Alvarid (2020).

In summary, the TDD follows the pattern of

1. write a failing test,
2. make the test pass,
3. refactor, and
4. continue in this loop (Freeman & Pryce, 2012).

Figure 28: TDD Cycle



Source: Alvarid (2020).

Behavior-Driven Development (BDD)

By using test-driven development (TDD) in projects in different environments, some confusion and misunderstanding may arise; programmers may not know where to start; what to test and what not to test; how much to test in one go; what to call their tests; and how to understand why a test fails (North, 2020). One approach to reduce such confusions is behavior-driven development (BDD), which is an agile software development practice introduced by Dan North (2006). BDD aims to provide a shared understanding of how an application should behave by discovering new features based on concrete examples. Those key examples are formulated using the natural language in a "given/when/then" structure.

- Given: Some initial context (the givens)
- When: An event occurs
- Then: Some outcome is ensured

An implementation of the "given/when/then" approach could resemble the following.

- Given: The user has entered valid login credentials.
- When: A user clicks on the login button.
- Then: The successful validation message is displayed.

Using such simple plain language (English here) helps all stakeholders (developers and clients) of the project to understand the feature behavior of a functionality. Gherkin is a domain-specific language for describing examples with "given/when/then" in plain text files, called feature files. Feature files describe how a system should interact with the user. A simple example of a feature file using Gherkin looks like

Code

```
Feature: Calculator

Scenario: “+” should add to current total
Given the current total is “5”
When I enter “7”
Then the current total should be “12”
```

The scenario in the example above describes a step in a “given-when-then” structure (Specflow, n.d.).

Acceptance Test-Driven Development (ATDD)

In the acceptance test-driven development (ATDD) method, a single test is developed from the perspective of the client with the focus on accurate functionality. Indeed, ATDD and BDD are very similar and the main difference is in their focus: ATDD focuses on the functionality and BDD focuses more on behavior. In this development method, different project stakeholders with different perspectives are involved in writing acceptance tests before implementing the relevant functionality. These acceptance tests will reflect the user’s perspective. The key differences between TDD, BDD, and ATDD are explained in the following table (Unadkat, 2020).

Table 2: TDD vs. BDD vs. ATDD

	TDD	BDD	ATDD
Definition	Is a development approach to implement a feature	Is a development approach based on the system behavior	Is a development approach for capturing the requirements (similar to BDD) by writing acceptance tests before implementing the relevant functionality
Main Focus	Unit tests	Understanding requirements based on system behavior	Writing acceptance tests
Participants	Developers	Developers, customers, quality assurance engineers	Developers, customers, quality assurance engineers
Language	Similar to the main code	Plain English	Plain English

Source: Alvarid (2020), based on Unadkat (2020).

We conclude this section with a brief discussion about pair programming, which is an agile software development technique in which two programmers work at one working station. In pair programming, one of the programmers, known as the driver, will write the code and the other developer, called the navigator, controls the code and concentrates on the plan of action. It has been found that for a development-time cost of about 15 percent, pair programming enhances design quality, technical skills, and team communications while also reducing defects and staffing risk (Cockburn & Williams, 2001). Pair programming was first introduced as a part of the extreme programming software development method (Beck, 1999). Extreme programming is a set of software techniques that focuses on exchanging information, clarity, response, determination, and consideration. The goal of extreme programming is to have a piece of firm programming knowledge (Unadkat, 2020).

3.3 Continuous Integration and Continuous Delivery

In this section, we focus on procedures for implementation and automation of continuous integration (CI), continuous delivery (CD), and continuous training (CT) of ML models. To achieve this goal, the MLOps principles should be applied to the ML project. MLOps is a machine learning development approach that aims to link ML system development (ML) and ML system operation (Ops). Applying MLOps requires a focus on automation and monitoring at all phases of ML system design, including integration, testing, release, deployment, and infrastructure management (Google, 2020b). We start this section with an introduction to continuous integration, continuous delivery, and continuous testing.

Continuous Integration (CI)

Continuous integration (CI) is a development philosophy in which developers commit their code to a version control repository periodically (normally teams have a minimal standard of committing code at least daily). The motive behind CI is to have a more straightforward method to recognize errors and other software quality problems on a smaller piece of code rather than larger chunks of code developed over a long period of time. Continuous integration was first introduced by Kent (Beck, 1999). As with other extreme programming practices, the intention behind continuous integration was that, if regular integration of the codebase is good, why not do it all the time? Before implementing CI into the software project, the following components are required (Humble & Farley, 2015).

- Version control: All elements in the software project should be checked into a version control repository like Git (Git, n.d.). This includes the source code, test scripts, database scripts, build and deployment scripts, configuration files, etc.
- An automated build: The build of the program should be able to run in an automated way via the command line. It can start with a command-line script to command the IDE (integrated development environment) to build the software. However, it could also be a complex combination of multistage build scripts that call one another.
- Agreement of the team: Continuous integration is teamwork (not a tool) and therefore it requires commitment and discipline from the members of the development team. Only by the adoption of the required disciplines in the team can continuous integration result in an improvement in product quality.

There are several CI tools such as GoCD (GoCD, n.d.), CruiseControl (CruiseControl, n.d.), and Jenkins (Jenkins, n.d.-a). After installing the CI tool, one can start the CI process by configuring the tool. The configuration will specify the location of the source control repository, the required scripts to run to compile the software, and the automated commit tests.

Continuous Testing (CTe)

As we discussed in the first section of this unit, automated testing assists the test engineers to write and run different types of the test during the software development lifecycle. The testing types range from a unit test to the system and regression test for the entire software (Sakolick, 2020). Regression testing is a type of testing in which you re-run the functional and non-functional tests on an already tested software product to check if the (modified) software passes the test after applying some new modifications (Basu, 2015). Regression tests follow other automated tests, such as performance tests, API tests, and security tests. All these tests are required to trigger through a command line or other automation tools. After automating the testing procedure, continuous testing (CTe) indicates that the automated test is already integrated into the CI/CD pipeline. In this case, some of the tests (like the unit and functionality tests) should have been integrated into the CI part (when CI identifies the issues before or during the integration process). Tests that need complete delivery conditions, such as performance and security tests, should usually be integrated into the CD. These tests are executed after the build process (Sakolick, 2020).

Continuous Delivery (CD)

Continuous delivery (CD) is a software engineering method in which project teams continually deliver valuable software in short cycles and guarantee that the software can be reliably delivered at any time (Chen, 2015). CD empowers companies to rapidly, efficiently, and reliably deliver service improvements to the market. During the development process, each development team uses one or more development and testing environments to stage the application modifications for testing. This process can be automated using the CI/CD tools such as Jenkins (Jenkins, n.d.-c), CircleCI (CircleCI, n.d.), Travis CI (Travis CI, n.d.).

A typical CD pipeline has the following steps (Chen, 2015).

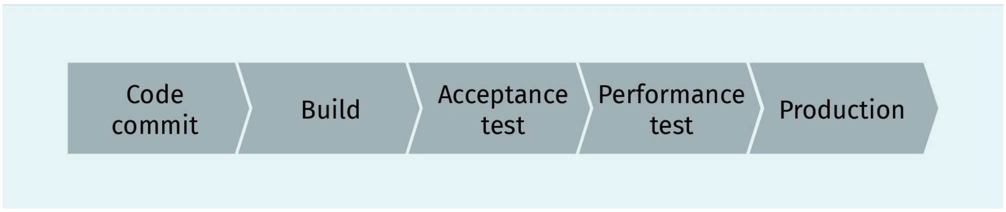
1. Code commit: The code commit step provides quick initial feedback to developers on the code they have already checked in. When a developer checks her or his code into the CI/CD tool, this step triggers automatically and executes the source code and the unit tests. In the case of an error, the pipeline halts and informs the developers about the problem. The developer resolves the issues and checks in the code. After re-running the code and in the case of an error-free execution, the pipeline moves forward to the next step.
2. Build: The build step performs the unit tests again to produce a **code coverage** report. In this step, the integration tests and various code analyses are executed, and artifacts are generated as output. These artifacts are uploaded into the repository that manages them for deployment or delivery. All the following pipeline steps will run using these artifacts.

Code coverage
This specifies how a developed software was successfully exercised during a test by utilizing metrics like function coverage or line coverage.

3. Acceptance test: This step essentially ensures that the software satisfies all defined user requirements. The pipeline generates the acceptance test environment in this step (a production-like environment in which the software is deployed). This includes provisioning and configuring the servers, deploying the software to the servers, and configuring the software. If the software passes the acceptance tests in this environment, the pipeline proceeds to the next step.
4. Performance test: In this step, the pipeline evaluates how the code modifications will influence the software's performance. The pipeline provides a suite of performance tests, and reports the results.
5. Production: The final step is deployment into the production environment.

All the aforementioned steps can be automated by utilizing CI/CD tools like Jenkins. In this case, developers could use Jenkins to describe a pipeline in a Jenkinsfile that contains different stages. The Jenkinsfile also includes environment variables, certifications, and other parameters (Jenkins, n.d.-c).

Figure 29: Continuous Delivery (CD) Pipeline



Source: Alvarid (2020).

Continuous Training (CT)

Continuous training (CT) is achieved when the ML pipeline is automated. To automate the ML pipeline by feeding new data into the model in production, one needs to implement the following steps into the ML pipeline (Google, 2020b).

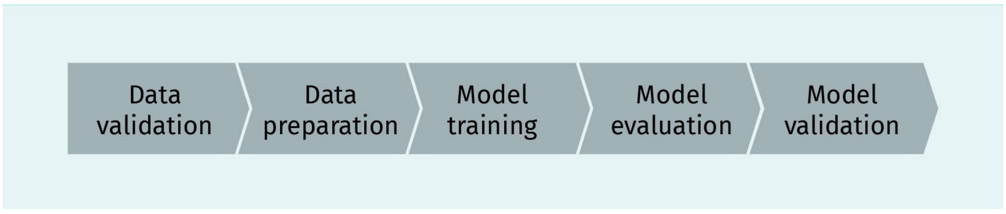
- Automated data validation: This step is performed before training the model to check whether the model should be trained or the pipeline execution should be stopped.
- Automated model validation: This step is performed using the new data right after the model training. This step is required to validate the model before going into production.
- Pipeline triggers: The pipeline could be triggered to digest new data by on-demand (manual) execution of the pipeline; on a schedule (daily, weekly, etc.); upon the availability of new data; a degradation in model performance; or a significant change in the data distribution.
- Metadata management: After each execution of the pipeline, some metadata—such as pipeline and component versions, the start and end time and date of the execution, the executor, or the configured parameters passed to the pipeline—should be stored in the metadata store of the pipeline.

A schematic representation of an automated ML pipeline for continuous training (CT) is shown in Google Cloud, found in Appendix 1 (Google, 2020b).

Here is the list of the ML pipeline characteristics for continuous training (Google, 2020b).

- Rapid experiment: The transition in pipeline experiment steps (data validation, data preparation, model training, model evaluation, and model validation) should be orchestrated to result in a rapid iteration of experiments.
- CT of the model in production: The ML model should be trained automatically in production using fresh data.
- Experimental-operational symmetry: A key aspect of an automated pipeline is to use the same pipeline from the development or experiment environment in the preproduction and production environment.
- Modularized code pipeline components: The components of the ML pipelines need to be reusable, composable, and potentially shareable across ML pipelines. Therefore, the source code for components must be modularized.

Figure 30: Orchestrated Experiment



Source: Alvarid (2020).

MLOps and ML Pipeline for CI/CD

An ML system is also a software system, therefore similar procedures apply for ensuring that ML systems can be built and operated reliably at scale. However, ML systems are distinguished from conventional software systems in a number of ways (Google, 2020b). For example, an ML project team needs data scientists with experience in explorative data analysis, model development, and experiments, but that doesn't necessarily mean that they have software development skills. The development process is different as well, as ML system development is experimental. This means a developer should try different algorithms, modeling techniques, and parameters to discover the best solution for the problem. In addition to the conventional tests—such as unit and integration tests—data validation and quality evaluation of the trained models should also be tested. The deployment of the ML systems can also differ in that it sometimes needs a multi-step pipeline which adds complexity to the deployment process. Finally, due to the evolving and dynamic nature of ML systems, there are more possibilities for model degradation than conventional software systems.

Due to the above characteristics of the ML systems, there are some differences in CI and CD with respect to the conventional software systems (Google, 2020b):

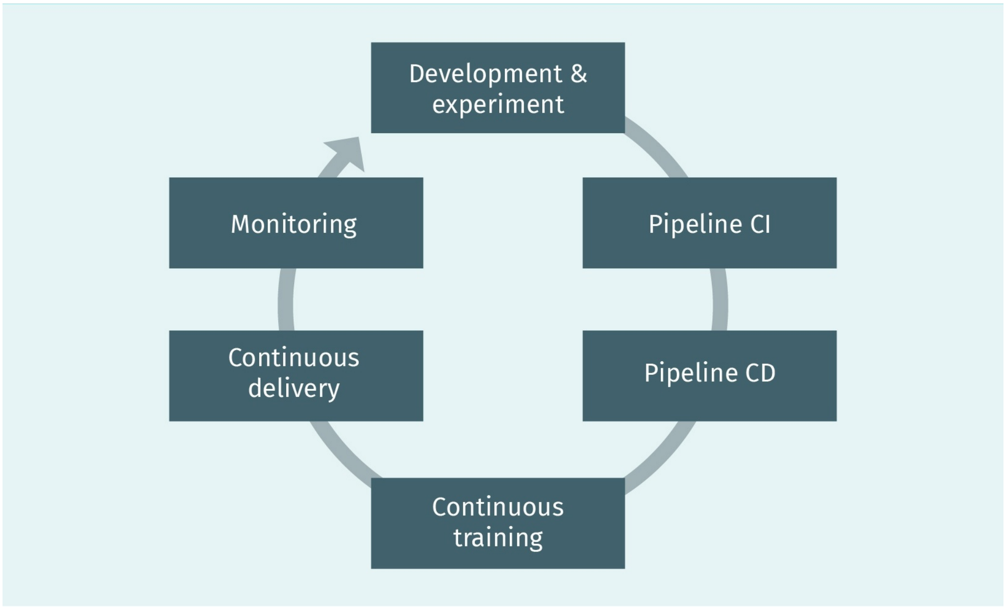
- In addition to testing and validating code and modules, CI in ML systems is also concerned with testing and validating data, data schemas, and models.

- In addition to a single software or a service, CD in ML systems is about a system (an ML training pipeline) that should automatically deploy other services (like model prediction service).
- CT (cntinuous training) is a new property that is unique to ML systems. CT is mainly concerned with automatically retraining and serving the models.

Considering the above differences, a CI/CD automation of an ML system could have the following six steps, which are also illustrated in the next figure (Google, 2020b).

1. Development and experiment: By examining different ML algorithms and new ML models, the development team finds the most appropriate source code for the ML pipeline which is pushed into the source repository as the output of this step.
2. Pipeline continuous integration: The source code from step 1 will be executed and tested. The output of this step is several pipeline components—such as packages, executables, and artifacts—to be used in later steps.
3. Pipeline continuous delivery: The artifacts generated in step 2 (CI) are deployed to the target environment. The output of this step is a deployed pipeline which includes the new ML model.
4. Automated triggering: The pipeline is automatically executed in the production environment in response to a trigger or based on a schedule. The output of this step is a trained model pushed into the model registry.
5. Model continuous delivery: The trained model is used as a prediction model to service the customers. The output of this step is the prediction service.
6. Monitoring: The statistics regarding the model performance based on the live data will be collected. The output of this step is a report and also likely a trigger to execute the pipeline for step 1.

Figure 31: CI/CD for an ML Model

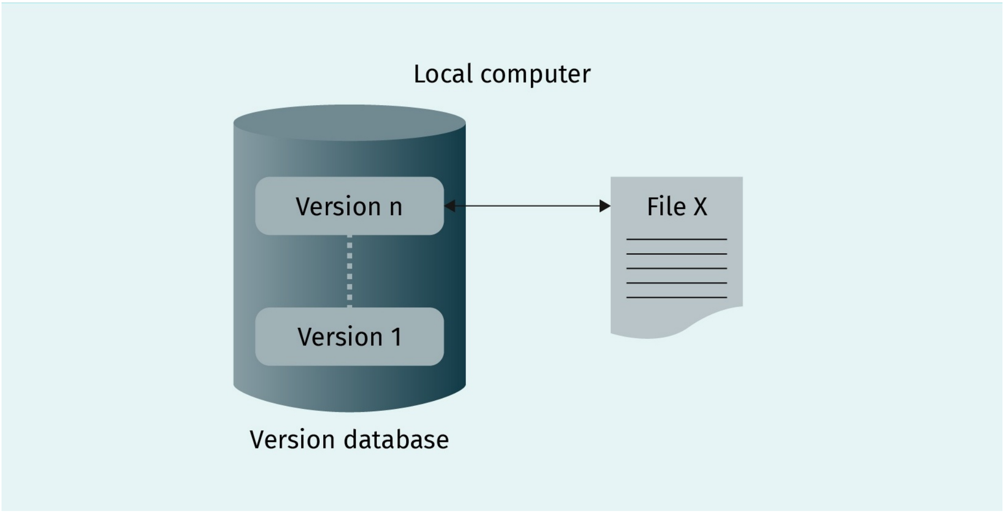


Source: Alvarid (2020).

3.4 Version Control

During the software development lifecycle, the development team must keep a historical track of every single change in applications, such as source code, project documents, or build script (especially in the case of a large software project with many developers and multiple teams). The response to this essential requirement has already been developed over decades: the version control system (VCS). A VCS also enables different developers and teams within a project to work together in parallel on separate parts of an application while keeping a system of records (Humble & Farley, 2015). Using a VCS, it is possible to revert selected files to a previous (saved) state or even revert the entire project to a previous state. The first version control system was developed by Marc J. Rochkind at Bell Labs in 1972 and was called the SCCS (Source Code Control System) (Rochkind, 1975). SCCS has been followed by many open-source version control systems like RCS (Tichy, 1982), CVS (Price, 2005), Apache Subversion (Free Software Directory, n.d.), and Git (Git, n.d.), as well as many commercial solutions like Perforce (Perforce, n.d.), StarTeam (StarTeam, n.d.), IBM Rational ClearCase (IBM, n.d.-b), Mercurial (Mercurial, n.d.), and Microsoft Teams Foundation System (MS Teams, n.d.). In this section, we will discuss one of the open-source VCS solutions. Because SCCS and RCS solutions are not very common these days, we won't discuss them here; instead, we will focus on one of the most popular VCSs: Git. Before discussing Git in more detail, we will have a brief look at different possible VCS categories. The first is local VCS. This is the simplest type of a VCS system which probably every developer has used at least once in the very early days of practicing coding. It is based on copying modified files into a new directory labeled with a date (perhaps a timestamped directory, if they're clever). Although it is the simplest approach for a single developer, it is also massively prone to errors like losing track of directories and overwriting files. To overcome this problem, one can develop a very simple local version database that keeps track of the changes.

Figure 32: Local Version Control System

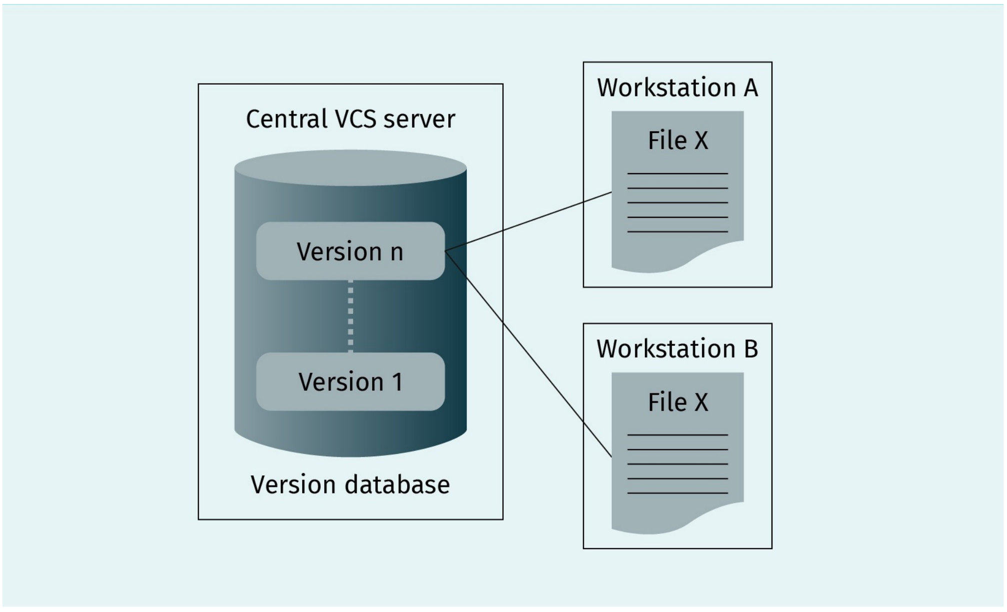


Source: Alvarid (2020), based on Git (2020).

The second is the centralized version control system (CVCS). If more than one developer or development working station is engaged in a project, another VCS solution should be used for this case, as shown the next figure. In a CVCS, the information regarding the file changes are stored on a central server and shared with the clients (workstations). Among the available products in this category, one can mention Subversion (Free Software Directory, n.d.), Perforce (Perforce, n.d.), and CSV (Price, 2005). This system has some benefits, e.g., every team member can see (as far as authorized) what other team members are doing and admins can control the roles. But there are some drawbacks such as a **single point of failure**, like a central server (if the server fails, workstations lose track of changes).

Single point of failure
Any non-redundant part of a system, the dysfunction of which causes the entire system to fail, is called a SPOF, or single point of failure (AVI Networks, n.d.).

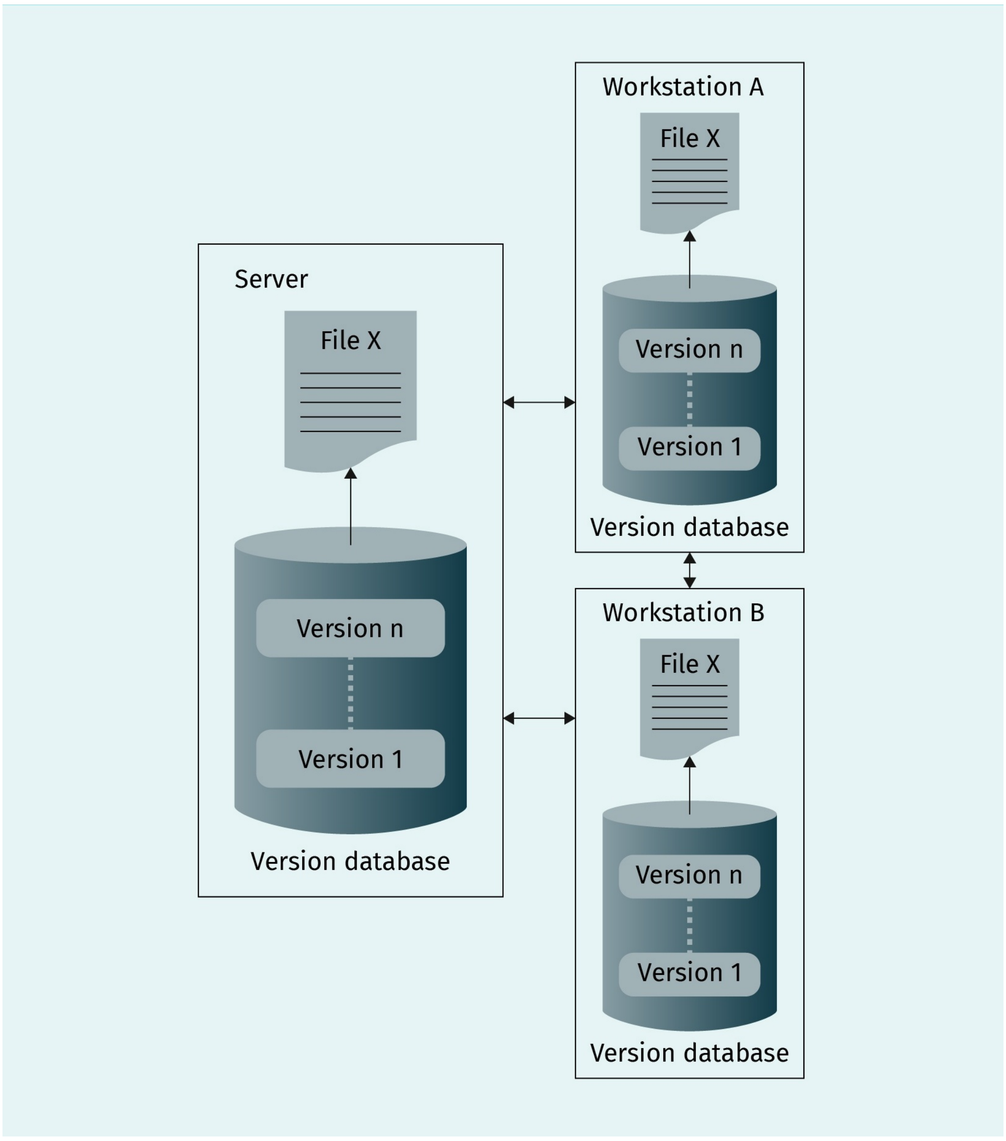
Figure 33: Centralized Version Control System



Source: Alvarid (2020), based on Git (2020).

Lastly, the ultimate solution to resolve the issues associated with the local and centralized VCSs is using a distributed version control system (DVCS), such as Git (Git, n.d.) or Mercurial (Mercurial, n.d.). In a DVCS, developers do not check out the latest version of the file, but they also fully mirror the repository. Therefore, the single point of failure problem is resolved because, in a DVCS, any single user’s repository can be copied back to the central server to restore lost data.

Figure 34: Distributed Version Control System



Source: Alvarid (2020), based on Git (2020).

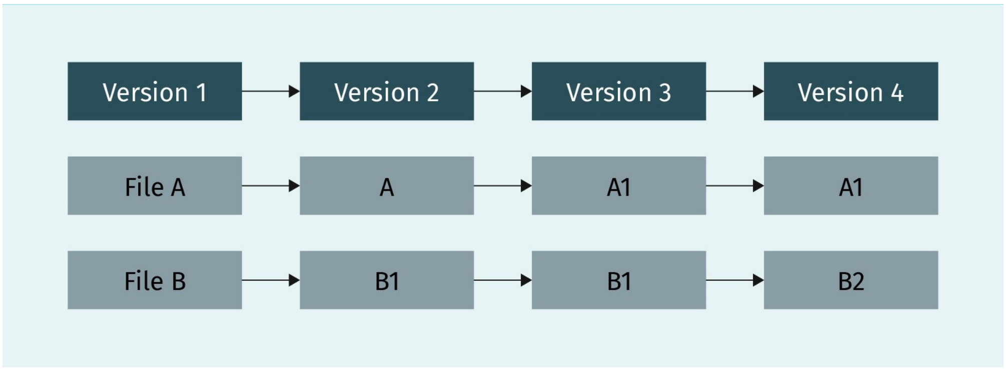
Git

Git project history is entangled with the Linux Kernel project, an open-source software project. In 2005 the Linux Kernel team decided to stop BitKeeper (a distributed VCS known as DVCS) as the version control system and develop its own VCS, a decision that led to the birth of the Git project (BitKeeper, n.d.). Git is a VCS that is simple but fast with strong support for non-linear development. It features thousands of parallel **branches**, fully distributed, and capable of managing large projects. Unlike most VCSs that store modification information as a list of file-based changes (i.e., storing a set of files and the related changes), Git stores a snapshot of every state of the project right after a user commits (sav-

Branch
A branch is a copy of the main repository of a version control system.

ing changes into the local repository). To be more efficient, if a file has not been changed, there is just a link to the previous state of the file in the new snapshot. In this regard, we could consider Git as a mini file system (Git, n.d.).

Figure 35: Git Snapshot Approach to Version Control



Source: Alvarid (2020), based on Git (2020).

One advantage of Git is checksum. For every change, a checksum is computed before it is stored, and it is impossible to modify anything without Git knowing about the modification. This feature prevents users from losing information in transit. To implement the checksum feature, Git uses SHA-1 hash, a 40-character string composed of the hexadecimal characters (Git, n.d.). This hash is calculated based on the file content or directory structure in Git.

There are three different states that a file in Git can have.

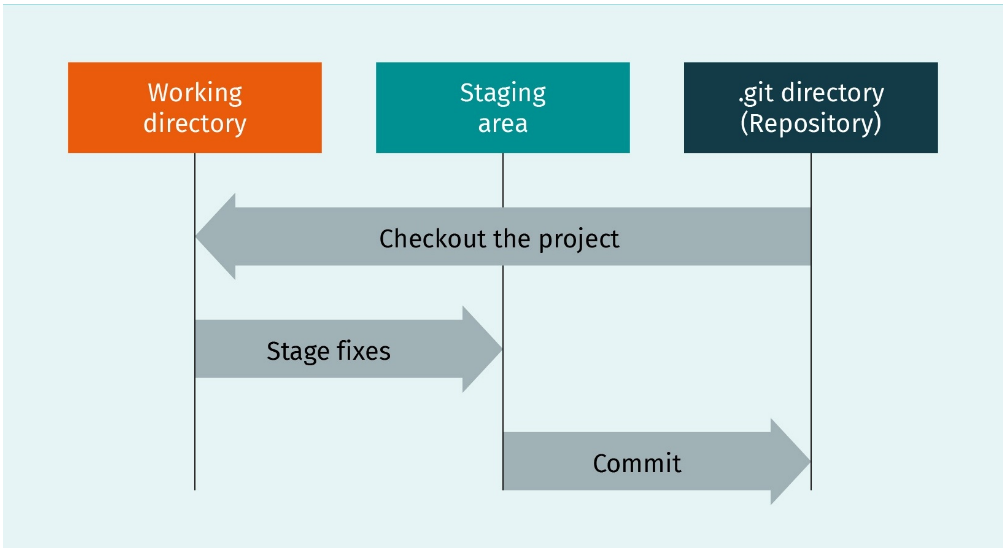
1. Modified: The file has been changed but the user has not committed the changes to the database. A file in this state is marked as `modified`.
2. Staged: A modified file has been marked by the user to go to the next commit snapshot. A file in this state is marked as `staged`.
3. Committed: The data have been saved on the local database safely. A file in this state is marked as `committed`.

Checkout
Checking out a branch updates the files in the working directory to match the version started in that branch, and it tells Git to record all new commits on that branch (Atlassian Bitbucket, n.d.).

Based on these three states, a Git project is composed of three main sections or stages. The first is the working directory, which is a single **checkout** of a version of the project. The files in this directory are pulled (incorporates changes from a remote repository into the current branch) from the Git repository. The second is the staging area. This is a file in the Git directory which contains information about the changes which are going to be committed in the next commit. Thirdly and finally, we have the Git directory. This is the most important part of Git that stores the metadata and object database of the project. When the user clones the repository from another workstation, it is the Git directory that is being copied. This process can be seen in the next figure. Regarding these sections of a Git project, a simple Git workflow has the following steps (Git, n.d.).

1. The user modifies the file in their working directory tracked by Git.
2. Then the user stages the part of the changes that should be committed in the next commit in the staging area.
3. The final step is to commit. This takes a snapshot of the files from the staging area and stores the snapshot in the Git directory.

Figure 36: Sections in a Git Project



Source: Git (2020).

Below, you can see a practical example of a Git Workflow (Dudler, n.d.).

1. Create a new repository in the project directory.

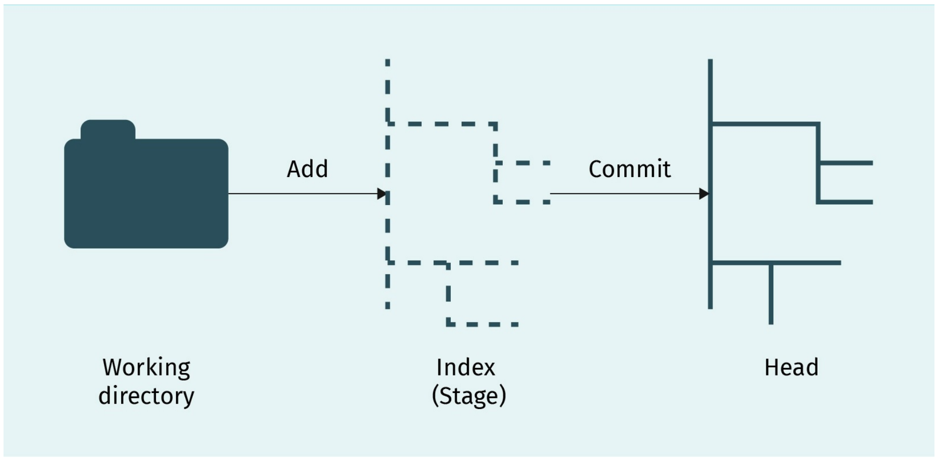
   Code
   ```
   git init
   ```

2. Create a working copy of a local repository by running the command.

   Code
   ```
   git clone /path/to/repository
   ```

3. The local repository consists of three “trees”:
   a) `Working Directory` contains the actual files.
   b) `Index` is the staging area.
   c) HEAD points to the last committed version.

Figure 37: Git Local Repository Trees



Source: Alvarid (2020).

4. Propose changes (add it to the Index).

   Code

   ```
   git add <filename>
   ```

5. Commit these changes into the Head.

   Code
   ```
   > git commit -m "Commit message"
   ```

6. Send changes to the remote repository.

   Code
   ```
   git push origin master
   ```

7. Create a new branch called “branch_1”.

   Code
   ```
   git checkout -b branch _1
   ```

8. Switch back to the master (main) branch.

   Code
   ```
   Git checkout master
   ```

9. Push the branch to the remote repository.

Code
```
Git push origin <branch>
```

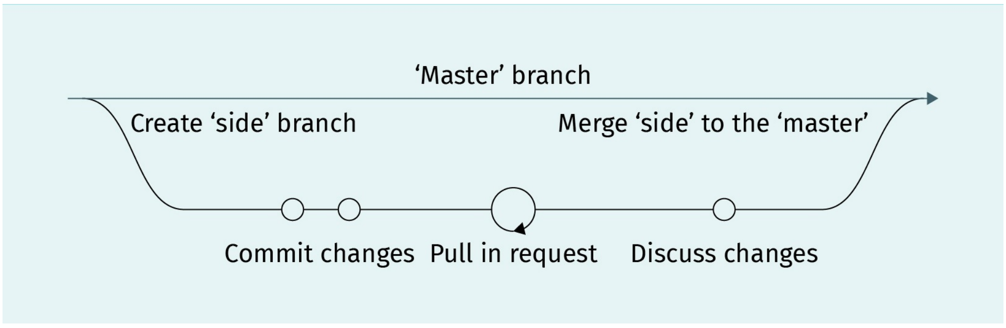
10. Update the local repository to the newest commit.

Code
```
Git pull
```

GitHub

GitHub is a cloud-based developer platform based on Git (GitHub, n.d.-a). The fundamental difference between Git and GitHub is that Git is an open-source tool where the developers can manage their source code locally, while GitHub is a cloud-based platform where developers can share their projects with a community, have access to developer tools and review each other's code, among other things. Like Git, at the core of a GitHub project is a project repository. A project repository contains all components that a project needs: files, folders, spreadsheets, data sets, etc. It normally also contains a README file, which contains information about the project. By default, the repository has one branch, called `master`. Branching is an approach to work on different versions of the repository at one time. The branches are used to test the changes in the source files before committing them to `master`. When a developer is working on a branch of the `master`, if there are some changes in the `master`, the developer of the branch could pull in the changes from the `master`.

Figure 38: Branching in GitHub



Source: Alvarid (2020), based on Git (2020).

3.5 Development Tools

Programming tools or software development tools are computer programs that support software developers and the development team to realize software projects. These tools are utilized to create and modify the source code of programs with text editors; get help in their program flow from special graphical user interface editors; translate their source code into an executable machine language with compilers or assemblers; test and debug developed solutions with test tools or debuggers; and store and manage programs and program documents with version control systems. In this section, we will review some of the most common development tools.

Command-Line Interface

The first, most straightforward, and most common development tool is the command-line interface (CLI). The command-line interface is a part of a computer program that receives a line of text as input from the user and interprets it (using the command line interpreter) in the context of a given operating system or programming language (Kumar, 2016). Operating systems and programming languages implement the command-line interface into a **shell** to access the operating system or write code. Examples of such CLI are Unix shell (Bourne, 1978), PowerShell (Bright, 2016), Z Shell (Z Shell, n.d.), and Python Shell (Python Shell, 2020).

Shell
In computer science, a shell is a software by which a user interacts with an operating system or program.

A command in an operating system CLI could have components such as

Code
```
prompt command parameter_1, …, parameter_n
```

Here, the `prompt` provides the context for the user (and usually ends with one of the `$`, `%`, `#`, `:`, > or – characters). The `command` is provided by the user to execute a special task, and `parameters` are the optional parameters provided by the client to control or limit the `command`. Although the CLI is the most straightforward way to communicate with an operating system or writing code, it is not the most suitable solution in some cases, especially for beginners, for in-line editing and for debugging. However, there are some integrated tools in each CLI which can make life easier for developers. There are many examples such as Vim text editor for Unix and Apple OS CLIs (Vim, n.d.); Wget, for retrieving files using HTTP, HTTPS, FTP, and FTPS via the web (Free Software Foundation, n.d.-a); and Gzip, a data compression program (Free Software Foundation, n.d.-b).

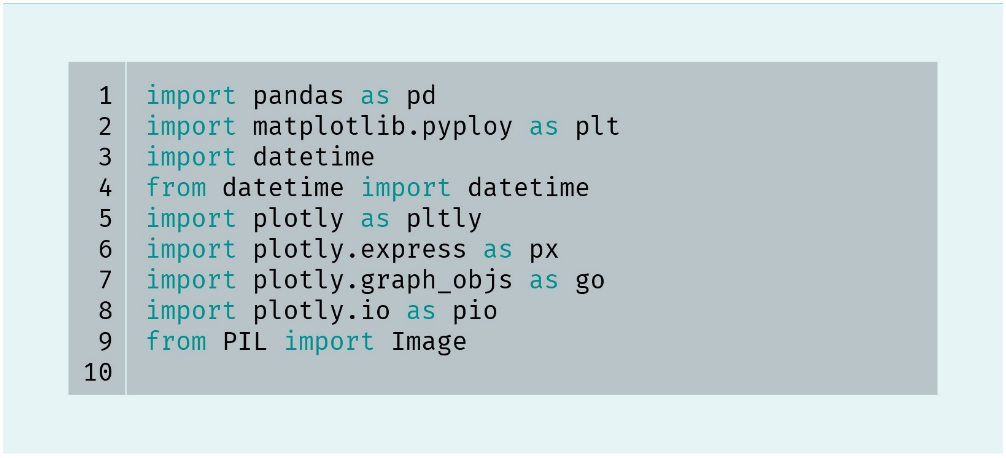
Integrated Development Environment (IDE)

An integrated development environment (IDE) is a single program or platform that comprises several developments that facilitate software development procedure. Examples of IDEs are Microsoft Visual Studio, Eclipse, NetBeans, and PyCharm. Typical functionalities of a modern IDE include

- intelligent code completion. This is a convenient method to access functions' descriptions and their parameter lists to speed up software development. In this method, after typing the name of a function in the IDE, a short description of the function together with a list of the input parameters of the function will appear. An example of such functionality is IntelliSense, which is a combination of code editing features including code completion, parameter info, quick info, and member lists (Visual Studio, n.d.).
- source code editor. The developer can use the IDE to write and edit the source code. Some features such as intelligent code completion, will facilitate the edition process.

- build automation. IDE can automate the build process, including compiling the source code, packaging the compiled filed into a compressed format, and producing installers.
- debugger. An IDE debugger lets you change the value of the variables at run-time, see the value of the variables also at run-time, break execution at any point in the code, etc.
- Syntax highlight. An IDE for the supported programming languages highlights the test with different colors and fonts as shown the following figure.

Figure 39: IDE Syntax Highlighting



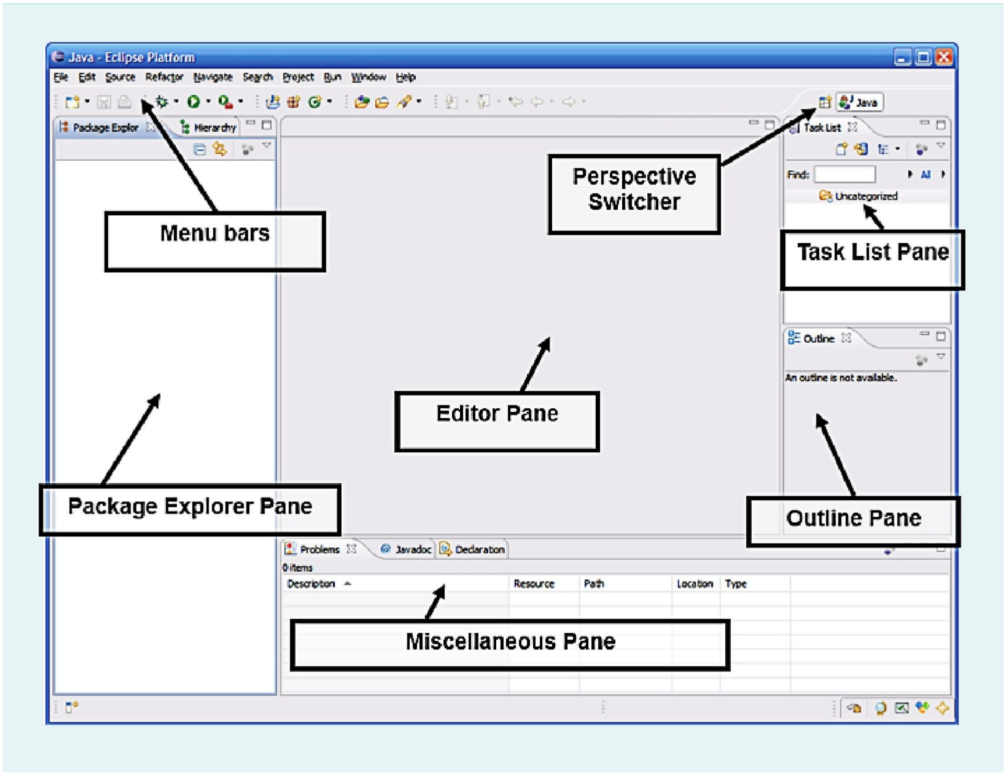
```python
import pandas as pd
import matplotlib.pyploy as plt
import datetime
from datetime import datetime
import plotly as pltly
import plotly.express as px
import plotly.graph_objs as go
import plotly.io as pio
from PIL import Image

```

Source: Alvarid (2020).

In the following, we will review one of the most common open-source IDEs: Eclipse (Eclipse, n.d.). Eclipse started as proprietary technology, led by IBM, and in 2001 the Eclipse open-source project was announced by the initial Eclipse Consortium (Eclipse, n.d.). It provides tools for coding, building, running, and debugging applications and was originally designed for Java. Eclipse now supports many other languages such as C, C++, Python, PHP, and Ruby (University of Maryland, 2018). The components of the Eclipse IDE are shown below.

Figure 40: Eclipse IDE Components



Source: Alvarid (2020), based on Hood (2018).

- Menu bars: full drop-down menus plus quick access to common functions
- Perspective switcher: to switch between various perspectives
- Package explorer pane: where our projects/files are listed
- Editor pane: the place to edit the source code
- Miscellaneous pane: several components are listed here such as a console and a list of compiler problems
- Task list pane: a list of “tasks” to complete
- Outline pane: a hierarchical view of a source file

Jupyter Notebook

Jupyter Notebook is a web-based interactive programming application that is used for developing, documenting, and executing code, as well as communicating the results (Jupyter Team, n.d.). Jupyter Notebook is mostly used by data scientists to run Python codes, but it supports about 40 different programming languages. In general, it can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter Notebook combines two functionalities. Firstly, it includes a web application (or a browser-based tool) for interactive management of documents that combines descriptive text, mathematics, computations, and their media output, such as diagrams or 3D-visualisations. The web application makes it possible to edit the text in a browser with automatic syntax highlighting; execute the code from the browser; represent the result of computations in PNG, SVG, HTML, and other rich media representations; and include mathematical notations using LaTeX. Secondly, it includes notebook documents which are a representation of all content visible in the web application. These include inputs and outputs of the calculations, descriptive text, mathematics, images, and media representations of objects, as well as the source code. Such documents are stored in a **JSON file** with the `.ipynb` extension. Storing as a JSON file facilitates the version control of the code. In addition, any `.ipynb` file which is available on a public URL can be shared easily (Jupyter Team, n.d.).

JSON files
These are pure text files that are only intended to exchange data between different applications.

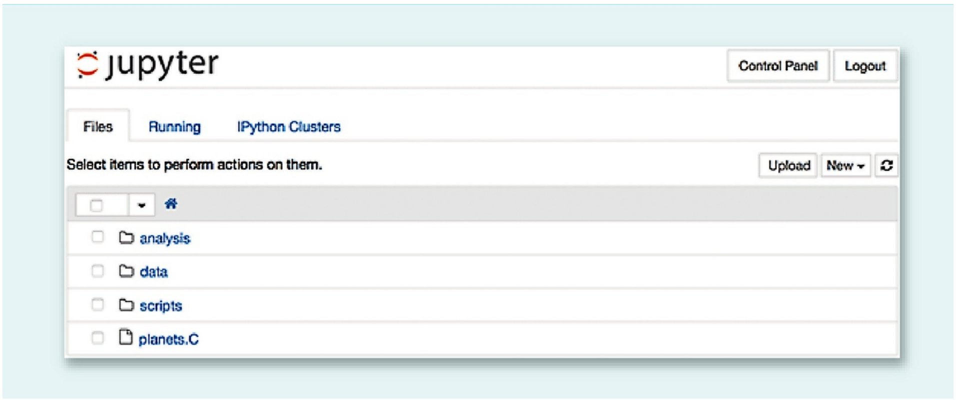
Using Jupyter Notebook

1. One can start a notebook server from the command line using the following command (Jupyter Team, n.d.):

   Code
   ```
   >jupyter notebook
   ```

2. The landing page of the Jupyter Notebook web application (by default, http://127.0.0.1:8888), which is called the dashboard, includes the available notebooks in the notebook directory.

   Figure 41: Jupyter Notebook Landing Page

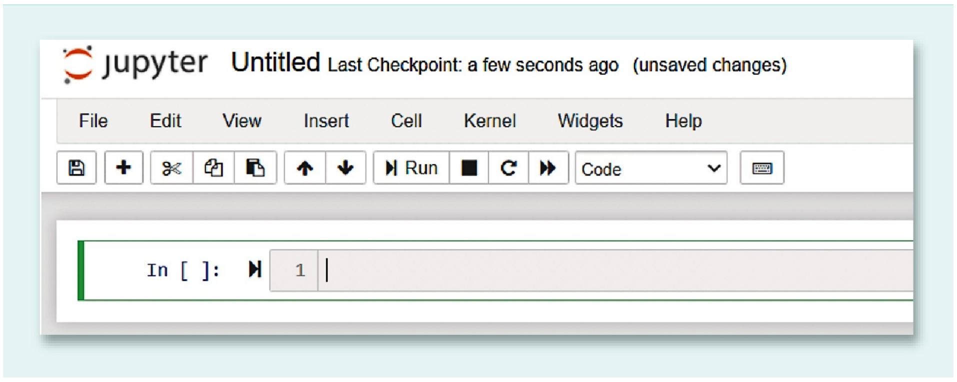


Select items to perform actions on them.
analysis
data
scripts
planets.C

   Source: Alvarid (2020).

3. The user can then create new notebooks from the dashboard under new/notebook. The new notebook (a `.ipynb` file) is shown in the next image.

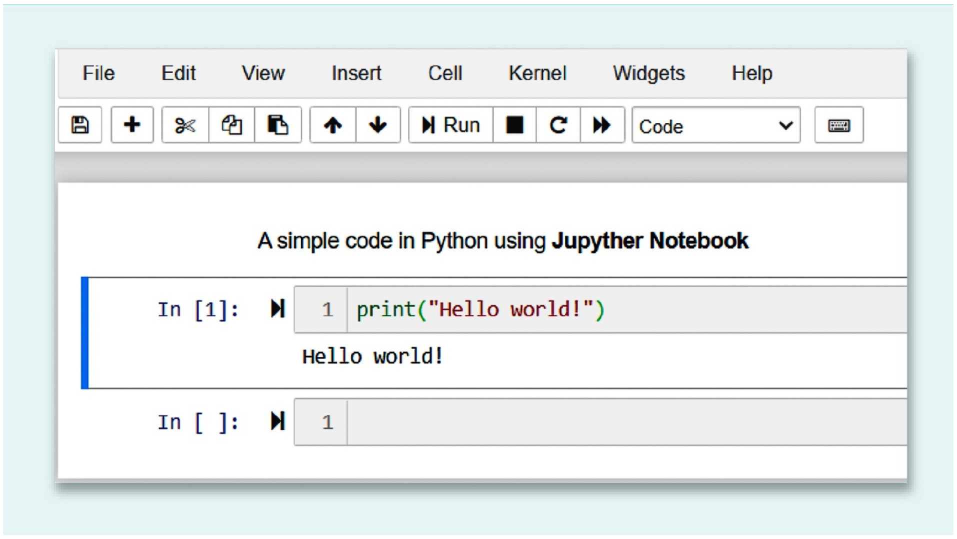
Figure 42: A Jupyter Notebook (.jpynb File)


jupyter
Untitled Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code
In [ ]:

Source: Alvarid (2020).

4. After this, the user can start to write code inside the notebook.

Figure 43: Using Jupyter Notebook to Run a Simple Python Code



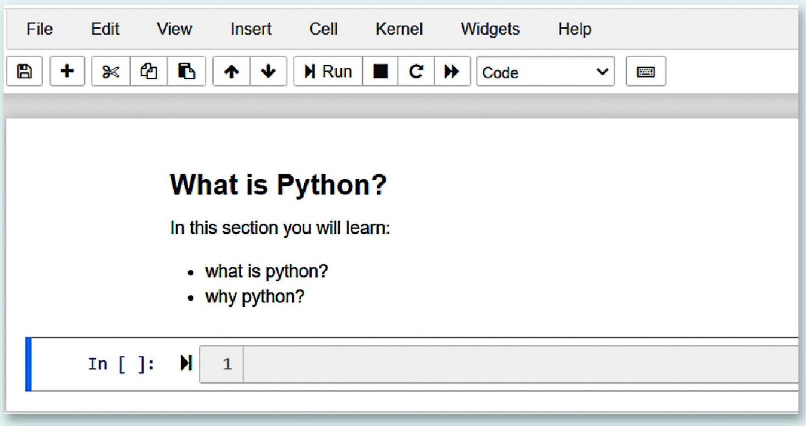
A simple code in Python using **Jupyther Notebook**

```
In [1]:  print("Hello world!")
Hello world!
In [ ]:
```

Source: Alvarid (2020).

5. Users can document the code in plain text using the markdown language. It also supports text markup (italics, bold, form lists, etc.) as shown in the following figure.

Figure 44: Using Markdown to Document the Code in Plain Text in a Jupyter Notebook



What is Python?

In this section you will learn:

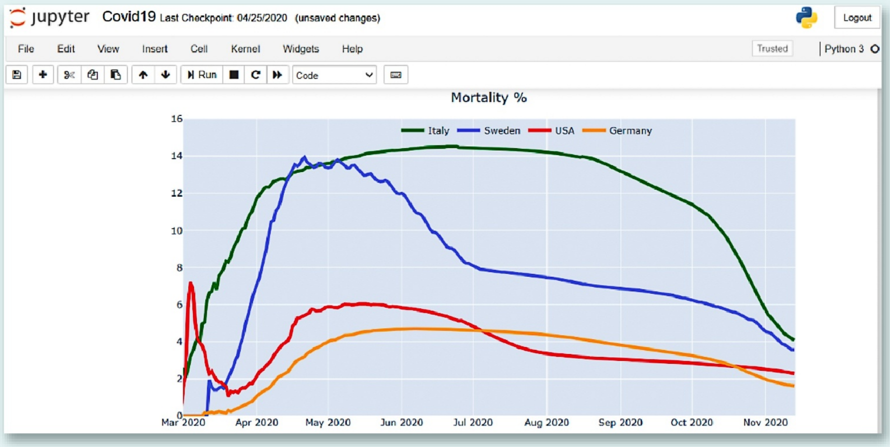
- what is python?
- why python?

```
In [ ]:    1
```

Source: Alvarid (2020).

6. One can also use Jupyter Notebook to plot in-line, as shown below.

Figure 45: In-line Visualization in Jupyter Notebook



Source: Alvarid (2020).

SUMMARY

In this unit, we discussed different approaches to software development with an emphasis on testing. We began by discussing different test scenarios such as unit, integration, system, functional, and non-functional tests (categorized by Marick's quadrant). Then we discussed how to test

a machine learning system as a non-deterministic model, as well as how both data and learning programs from ML models should be tested. Three approaches to test in parallel to developing were introduced: test-driven development (TDD), behavior-driven development (BDD), and acceptance test-driven development (ATDD). After this, we learned how to automate the pipeline of project development using continuous integration (CI), continuous delivery (CD), continuous testing (CTe), and continuous training (CT) in the case of machine learning pipelines. For the CI/CD in a machine learning pipeline, we learned that there are six steps: development and experiment, pipeline CI, pipeline CD, continuous training, continuous delivery, and monitoring. We were then introduced to the principles of version control systems for the software projects and explained two common solutions: Git and GitHub. Finally, we learned about the development tools such as command-line interfaces (CLI) and integrated development environments (IDE).

UNIT 4

API

STUDY GOALS

On completion of this unit, you will have learned …

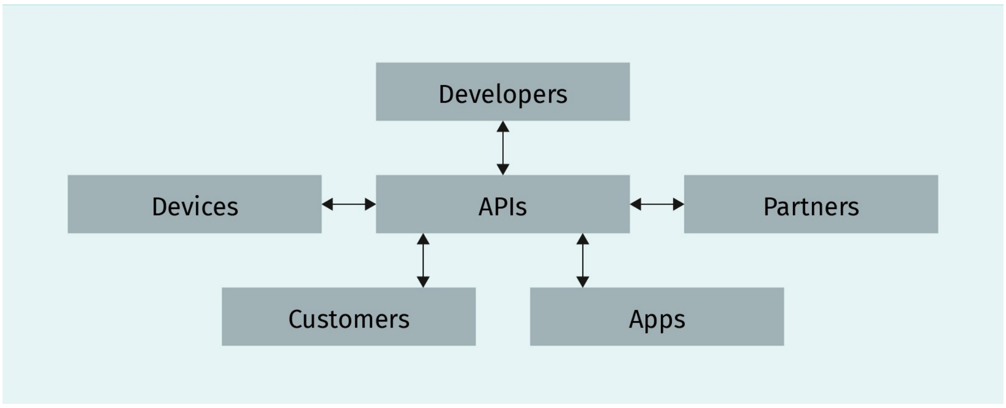
- the various ways of interacting with software and services.
- the most important principles to know when designing and building interfaces.
- how to distinguish good from bad interface design.
- how to build a Python library with good design in mind.

4. API

Introduction

In an increasingly digitalized and interconnected world, a large part of the world population has access to smart devices and uses them for an ever-growing range of use cases. That means that even technically inexperienced or inapt users cannot help but use complex software on a regular basis. To make this work, the ways in which you interact with your computer must be well-designed and easy to grasp. Not only that, the interface you're using to interact with any piece of software must be stable and reliable, so that the services you're enjoying don't surprise you or suddenly stop working. Take Amazon's retail business as an example. Chances are you've seen an e-commerce platform like Amazon, browsed for products, selected a few of them and put them in your cart. The website you intuitively interact with is just the user interface to a massively complex software and hardware backend powered by tens of thousands of machines scattered around the globe and operated and maintained by an army of engineers who built that platform in the last two decades. Since it's so common today to use services and check apps, it's also easy to forget about the typical discrepancy between the way you interact with software and the rest of it. Your interaction with apps can be seen as the figurative tip of the iceberg of software complexity, and while some websites might be overwhelming, or simply poorly designed, you can rest assured that the interface you're using is as much about what you need to know as it is about all the things you mustn't be exposed to. However, the story doesn't end at the level of clicking through user interfaces. All software engineers and data scientists interact with computers every day, and while their interaction looks distinctively different from that of your average browser user, they nevertheless still have to rely on contracts to interoperate with their programs and tools: These contracts are called application programming interfaces (APIs). The term API is increasingly known by non-professional tech enthusiasts, and, to stick with the example of Amazon: Exposing their internally developed tech stack through APIs in Amazon Web Services (AWS) has become a business line of the same order of magnitude as their retail endeavor. While not every click on a website is an API call, every business-relevant action a user carries out corresponds to one. In fact, one can take an API-centric view of modern, digitized business interactions, in which not only API developers, but also apps and devices, as well as your customers and strategic partners, all interact through APIs.

Figure 46: API-Centric View



Source: Pumperla (2020).

This unit is devoted to giving you an overview of common API paradigms, that is, the multitude of ways to interact with computers in a productive way. The focus for this unit is very much on a specific group of users, namely engineers and scientists, power users like you. Apart from being aware of what you're using and how you're using it, as an experienced professional it is crucial to develop a mindset of evaluating interfaces critically. In the end, a lot of the code you produce on the job will be an API for someone else to consume, so it's worthwhile to develop the skillset for building great interfaces yourself. Being on the producing or consuming end of an API is just two sides of the same medal. Therefore, we're going to build out a few examples of APIs in Python to demonstrate what good interface design looks like in practice, while also showing you counterexamples to learn from. As Martin Fowler puts it: "[a]ny fool can write code that a computer can understand. Good programmers write code that humans can understand" (Fowler & Beck, 1999, p. 15).

4.1 Interacting with Software and Services

Modern computers have an incredibly high level of complexity. Even if you consider yourself an expert, the only reason you're capable of being productive in your interaction with your machine is that you rely on the existence of many layers of abstractions that smartly hide what's going on under the hood. For instance, if you're doing a data analysis in Python, you want to focus on your specific use case and not how your code translates into zeros and ones, or how exactly your CPU is utilized by your operating system to run this analysis. Ultimately you have to trust that your code, a tiny fraction of all programming capabilities, does what you intend it to do. In other words, you're constantly using application programming interfaces (APIs) to shed complexity and be productive. As alluded to in the introduction, an API is an interface that allows the interaction between one or several parties and a software component. To be more precise, it is key that an API

- defines the requests or calls you can make with it,

- clearly states the data formats it expects as input and what formats it produces, and
- communicates which conventions it follows and what the intended behavior of each request is.

To give you a first, concrete example of a class of API calls, consider Python's functionality to import libraries. It is centered around a single keyword: `import`. There are three essential calls you can do with `import`, namely, importing Python modules, importing functionality from a module, and aliasing. Throughout this unit we're using Python 3.7 or later for our examples. Here's an example:

Code
```
import time # import the time module
from time import ctime # import the “current time”
import time as t # refer to time module as t
```

To use `import`, you always have to follow it up with a valid Python package name, i.e., either a built-in package or a third-party package installed in your Python environment. An `import` statement does not return anything if successful (we say it is silent), but fails with an error message if the imported library does not exist. For instance, if you type `import testfailure` into a Python session, you will see the following error message:

Code
```
ModuleNotFoundError: No module named ' testfailure '
```

In terms of intended behavior, unsurprisingly, the role of `import` is to import and make Python functionality accessible to users in their current session or script execution. For example, the above import statements would allow a user to query the current system time with `ctime()`, which is returned to the user as a Python string. Before we dive into more aspects of APIs and examples of types relevant for your work and studies, let's have a look at the evolution of the term API and what it means today.

Historical Evolution of APIs

The term API has undergone an interesting evolution since its inception. From today's perspective, programmers have arguably been building APIs ever since the first programs have been written. Originally, “API” was only used for user-facing applications, but quickly expanded from there. The term itself was formally published in the 60s (Cotton & Greatorex, 1968), but it took some time before the concept was popularized. Cotton and Greatorex (1968) proposed a consistent application interface written in Fortran, intended to make the graphical application they were concerned with easier to use by making it run on any kind of hardware. Hardware independence is crucial to distribute your software across many different machines, something that is obvious to modern application designers but was difficult to achieve in the early days of computing. In 1974, the term API was expanded to database management systems, but the application interface was strictly separated from other ways of interacting with your database, such as querying (Date, 2019). This turned out to be too strict, and later generations realized that by expanding your definition of an application interface, you could unify all sorts of interactions. Building rich, integrated interfaces thus became the norm throughout all fields of computing and not only in database design (Berg et al., 1981). By the 90s, it was universally accepted that the term API would be more broadly used to cover all kinds of programming and defined as "a set of services available to a programmer for performing certain tasks" (Malamud, 1990, p. 294). With the inception of the internet, web APIs became a new extension of the term "network-based Application Programming Interfaces" (Fielding, 2000). Web APIs are so common today that people use "API," historically a somewhat restrictive term, to refer to web API and their communication protocols. Most of the time it is understood in the even narrower sense of JSON or XML-based web APIs. Throughout this chapter we take a more holistic view on application programming interfaces, of which web APIs are just a part. The quick adoption of the internet and increased interconnectivity of users throughout the world led to an explosion of web APIs which shows no sign of slowing down. But their beginnings were modest. Early pioneers of commercial use of web APIs include dominant forces from the dot-com era, including Salesforce, Ebay, and Amazon. For instance, in 2002 Amazon launched its Web Services (AWS), which allowed developers to embed content from Amazon into their own websites. AWS has vastly evolved since then and Amazon's cloud offering is almost on an equal footing with its retail business. The next wave of web APIs, starting around 2004, brought us social media integrations with tech giants such as Facebook, Twitter, or Flickr. Twitter published its developer API in 2006 to give engineers access to all kinds of data from the platform. In the same year, Facebook launched its API to let developers access friend information, photos, posts, etc., which had a high impact on attracting advertising companies and helped Facebook become a global player in the social media industry (Lane, 2019). In yet another wave, companies began to move their infrastructure and services to the cloud, pioneered by Amazon in 2006 with a new batch of services, led by Amazon Simple Storage Service (S3) and Amazon Elastic Cloud Compute (EC2), both of which are still elemental to AWS's success story. While S3 made cloud storage accessible with a simple pay-as-you-go approach, EC2 gave developers access to compute resources in the cloud, spinning up their own machines and hosting their services there. It's interesting to acknowledge the fact that the success of AWS is partially credited to Amazon's API-oriented culture (API Evangelist, 2012). Internally, to this day, all Amazon teams have to expose their functionality as services for other teams to use. While this philosophy might have incurred some overhead in the short-term, as building APIs is expensive, the genius move of exposing the company's infrastructure as a service to the public would not have been possible without it. With the release of the iPhone, the internet has become more mobile and so did the next generation of APIs. To briefly mention a last notable shift in the tech landscape that had an impact on APIs, with the emergence of the Internet of Things (IoT), more and more devices are now connected to the internet and eligible for API consumption. Examples include Amazon's Alexa (2014), which allows API access for each notable feature, such as voice interaction, alarm setting, or providing weather information. Fitbit (2017) produces fitness-metric-tracking wearables powered by APIs. In a historical context, nothing speaks more for the importance of APIs and the commercial interests they represent in this modern era than the infamous case of tech giants Oracle and Google fighting over copyright issues of 37 Java APIs in 2017 (Tsidulko, 2020). To conclude, today's APIs and their usage are ubiquitous and found in many areas relevant to you:

- APIs are behind practically all web applications.
- They power mobile and desktop applications.

- APIs underlie all network communication between the devices and applications on the internet.
- They are a cornerstone of the Internet of Things (IoT), connecting smart devices such as automobiles, fridges, or vacuum cleaners.

Specifications and Contracts

Every piece of software has a provider, the company or group of people who authored it, built it, and currently maintain it, and consumers, who utilize the software to cater to their needs. Consumers can either be humans or other machines using the software. Regardless of the actual roles, an API can be seen as a specification or contract between providers and consumers. As with any other contract, the parties involved must first come to an agreement regarding what the contract is about and then write it down. In software, the provider can dictate what an interface looks like to a certain degree but must make sure not to lose its customer base, for instance by sticking to bad design. By the time an API, which is viewed as a contract, is agreed upon, it is binding for all parties. Companies providing software-as-a-service (SaaS) usually have service level agreements (SLAs) that include API stability. In particular, a company can't change the structure of its API calls over night. This perspective implies certain problems when it comes to the maintenance and extendibility of APIs. Not unlike software packages or operating systems, APIs come with versions attached to them. For instance, the developer platform GitHub has a developer API, the URL of which reads

Code
```
https://developer.github.com/v3/
```

Note the `v3` suffix in the URL: This is the current stable version of GitHub's API, but historically it is the third major version of their API. The first two have been deprecated and removed years ago. The benefits of API versioning are manifold:

- Within one version you guarantee that every single API call has a stable interface. The consumers of the API building downstream applications can rely on it to not stop working without fair warning.
- A new functionality can be added to an API version if it is independent of old requests. In software releases this would correspond to minor releases or patches. Although bug fixes are part of keeping an API alive, this type of API extensibility can be very convenient within a release.
- New API versions are reserved for fundamental changes. We speak of breaking changes if code that consumes an API in an older version stops working with the new one. API developers should try to get interfaces right first-shot, but of course that's not always possible. It is still important to think about user-facing API design decisions carefully, as they are more difficult to fix than internal processes. Versioning APIs allows providers to roll out several versions of their API at the same time and gives consumers a fair warning of how long the old version will still be alive and how to migrate gracefully to the latest. Giving users time to react to changes and adapt is a massive benefit as opposed to making abrupt changes over night.
- Communicating **deprecation** warnings for older API versions while still maintaining them is an added burden on providers, but it is much more convenient for consumers.

- Old API versions can be deprecated when they become buggy, highly inefficient, or encourage bad coding practices. Individual calls can be removed entirely in a new version if they're not needed anymore.

Deprecation
An API can declare some of its calls as deprecated, which means that this part of the API is either modified or removed. Deprecation warnings give developers the chance to transition to parts of the API with long-term support.

Documentation

A topic that deserves special attention in the context of API contracts is documentation. While often seen as an unwanted but necessary evil on the part of application developers who want to focus on their code, documentation is crucial to enforce a common understanding of an API. In fact, no matter how simple you think an interface is, in the end a human will have to operate it (if only to instruct a machine to use it) and there's always a chance for miscommunication. Some go as far as to say that the documentation is the API and the code just its implementation. From a software planning perspective this is not an unusual thought. Before the advent of the Agile Manifesto, it was customary in all engineering fields to write explicit, extremely detailed specifications (or specs) that had to be adhered to before any of the coding or building could begin. A spec—for an API or another product—can be seen as the result of a potentially long negotiation process that in project management is often described as requirements engineering. From a pragmatic point of view documentation is not to be underestimated. In an API variant of Murphy's Law, you could say that any API call that can be misunderstood will be misused, so it's worth being as clear and precise as you can when building and documenting interfaces.

Security, Governance, and Release Policies

APIs allow companies to integrate with technology outside of their corporation, which introduces dependency. Likewise, when a company offers access to an API, it suddenly must deal with external dependents. Both sides of this coin do not only imply governance issues, but also potential security risks. On the governance side, if your business relies on the availability of an API and that API slows down, it will slow down your business, too. So, the performance monitoring of that external API becomes an integral part of your internal business processes. On top of that, if an integration suddenly stops working, you must figure out quickly whether it's due to a change on your side or from the external provider. That means that your internal development cycle must consider the external API development cycle that you rely on. It is not uncommon for larger companies to have a dedicated API manager who plans and manages all (external) API-related topics within a company, such as making sure the integrations comply with organizational or governmental standards. On the flip side, if you're exposing an API to the public, chances are that this API accesses production data of your company, which can potentially be compromised. For instance, as a health care data provider in most countries, you must keep your data anonymized at all costs. The security risks for companies with sensitive data can be very high, so investing in sufficient security standards and setting proper access rights for your APIs is crucial. On a high level, there are three main policies which APIs can be released under (Boyd, 2014).

- Private: Your API is only used internally.
- Partner: You expose your API only to a specific group of users, namely, specific business partners.

- Public: Your API is publicly available. In typed programming languages you often encounter the keyword "public" for a class or function, which means that this part of the respective library is available to all programmers working with that library. However, this kind of "public" functionality can still be part of a private API inside a company. Note that "**public**" is not the same as free or available to everyone, it just means that if you're granted access and pay for it, you can use it (see Fowler, 2002).

Published vs. public interfaces
Only if you publish an API, as in you expose it publicly on the internet, does it truly become public in the sense discussed in this unit, with all benefits and drawbacks.

One of the advantages of versioned APIs we discussed was interface stability within one release of an API. A common practice among API designers to grant them some flexibility is to mark their calls as unstable. Google's Java library Guava marks unstable components with a `@Beta` annotation, indicating that you can't rely on it, as it might change soon or vanish altogether.

API Types

Now that you've seen some examples, learned about the history of APIs, and understood some of the key aspects of what makes and breaks an API, let's have a look at various levels and types of APIs. You will notice that we're working "inside-out" from the lower-level components of a computer to higher-level application design. This onion-like layering of API abstractions is what makes it possible for you to work efficiently with a computer. Writing a script like the one you're reading right now was done on a text processing system without thinking for a second about the hardware that's running it.

Operating systems

Simply put, an operating system (OS) is software that manages and orchestrates hardware components and software resources. It also provides common system-level services for computer programs run by its users. The **POSIX** standard originated from work by Richard Stallman in the 1980s to unify different UNIX operating systems under one umbrella and is the foundation of the Linux operating system (Stallman, 2011). POSIX specifies a set of APIs that POSIX-compliant operating systems have to implement, so that compiled programs based on these APIs can run on any other system running POSIX. Some POSIX aspects you probably interact with daily when using a Linux system are its standard for files and directories, pipes, and specifications for input and output (I/O Port Interface and Control). Operating systems are very complex pieces of software, and there is a lot more to say about them, in particular about the topic of APIs, but that's beyond the scope of this script. Other examples of notable APIs that work close to the operating system level and are relevant for data-intensive sciences include

POSIX
The portable operating system interface, or POSIX, is a set of standards specified by the IEEE Computer Society. POSIX defines an API on the operating system level, including pipelines, processes, shells, and utilities, among many others.

- OpenGL (Open Graphics Library), a cross-language and cross-platform API for hardware-accelerated rendering of 2D and 3D vector graphics, which is extensively used in video game development, computer-aided design (CAD), and virtual reality applications. First released in 1992, the API standard is now maintained by non-profit consortium Khronos Group.
- OpenCL (Open Computing Language), a framework and API for writing applications independently from specific platforms and for leveraging various hardware components such as central processing units (CPUs), graphical processing units (GPUs), and field-programmable gate arrays (FPGAs). OpenCL provides C/C++-based programming

languages to users and comes with API calls for parallel processing on devices. Like OpenGL, OpenCL is maintained by the Khronos Group consortium. There are OpenCL-compliant implementations for hardware of all major hardware providers, except for Apple, who deprecated OpenCL support in favor of another standard.
- CUDA (compute unified device architecture), an API for parallel computing on GPUs developed by Nvidia for its own hardware. CUDA works for instance with C/C++ or Fortran, and while not trivial to program, it is considerably more convenient when compared to OpenGL. Efficient and fast implementations of matrix operations, such as Matrix multiplications and higher-dimensional equivalents thereof have been one of the core drivers behind recent successes in the field of deep learning.

Command line interfaces

If you're regularly using UNIX systems such as Linux or MacOS, but also if you're a Windows power user, you will likely execute a fair share of your programs in a command line shell or just **shell**. Tools run on the shell are referred to as command-line interfaces (CLIs) and are a special type of API leveraging OS resources. CLIs are simple yet powerful text-based systems that read instructions line-by-line. Among the many CLI tools available on various operating systems, two tools deserve special attention for Python programmers, namely, `python` itself and the package manager `pip`. While you can start an interactive Python session by just typing `python` into a shell on a system that has Python installed, you can also use it to start Python programs or execute strings interpreted as Python code directly from the shell. Here's an example of printing the number 42 as output to your shell using the `python` CLI:

Shell
A shell is a command-line interpreter which has access to the API of your operating system. While it refers to graphical interfaces as well, the term shell is mostly used to refer to command-line interfaces (CLIs). The naming convention comes from the fact that shells represent the outer layer of an OS.

Code
```
python -c "print(42)"
```

The `-c` flag allows you to pass any string that can be read by the Python interpreter. Another example of using `python` as a CLI API is to query the version installed on the system with `python -V`. To install packages for Python the standard way is to leverage the Python Package Index (PyPI) via the CLI tool `pip` (although there are other tools, like the old `easy_install`, the fashionable `poetry`, or the batteries-included `conda`). To install the popular pandas library for data analysis you can use

Code
```
pip install pandas
```

Updating pip itself, which is just a Python package on PyPI, works by invoking

Code
```
pip install --upgrade pip
```

As an aside, note that in the context of web APIs you often hear developers talk about having an API and a CLI for a given service. In that case, "API" usually refers to some sort of web interface or access through a programming language, while the "CLI" is the part of the API that is accessed through a shell.

Programming languages

We have just seen how Python can be used as a CLI tool to execute commands in a shell, but of course it is a full-blown programming language, and one that should be universally praised for its elegant design and high usability. We're not going to talk about any Python language specifics here, but note that the full set of data types, variables, and built-in functionality and classes of the Python language is in fact an API for users to interact with. A lesser-known fact to beginners is that the Python API has many different implementations. When you say you're programming Python, chances are that you're referring to CPython, a ubiquitous, free, and open-source implementation of the Python API based on the C programming language. However, there are alternatives available, such as Jython (written in Java), PyPy (written in RPython), and IronPython (written in C#). The take-away here is that while working with any given programming language it's easy to forget that they're all implementations of a design specification, written in another programming language. Python is a very expressive language, but in the end it's just an API for lower-level code. You can use this knowledge to your advantage too, since working with CPython allows you to write extensions of Python written in C, which might just be faster than plain Python. Popular libraries such as numpy, a Python package for manipulating n-dimensional arrays efficiently, is by and large written as a C-extension to Python.

Programming libraries and frameworks

Stepping out another layer, the libraries and frameworks you're using in practice—for instance libraries for data analyses or visualization—are themselves extensions of the programming languages they're written in. We mentioned numpy in the last paragraph, which is an API for n-dimensional arrays expressed in Python. The key point is that you don't have to think or reason about how numpy does what it does; you only need to understand the interface it is providing you with on a conceptual level. In other words, the interface is separated from the implementation details. This crucial principle underpinning all APIs is something we'll investigate more closely in the next section. For now, let's start with a simple example: Let's say I want to create a 3-by-3 matrix in Python, add this matrix to itself, and then print the result. `numpy` allows you to do that as follows:

Code
```
import numpy as np
x = np.ones((3, 3))
print(x + x)
```

This returns the expected result

Code
```
[[2. 2. 2.]
[2. 2. 2.]
[2. 2. 2.]]
```

The only thing you need to know is that the numpy API has a `ones` function call that creates matrices containing only the number 1 in all entries, of the specified shape (here 3 by 3), and that you can add matrices by using the "+" operator, which Python allows you to override. Crucially, you do not need to know how numpy stores matrices, or how it carries out matrix addition or any other operation. This decreases cognitive load and lets you focus on the use case you care about by only learning the API, not the implementation. Highly specialized libraries or frameworks can sometimes be characterized as domain-specific languages (DSL) when they act as a de-facto standard and complete toolset for a concrete domain. Numpy is a DSL for n-dimensional arrays in Python, while pandas would be an example of a DSL for data frames in Python. Becoming an expert in any data-intensive science usually involves having the essential libraries for your domain under your belt.

Databases

Another important application domain that we already briefly touched in the historical timeline of APIs is that of databases. Database management systems are complex software components that users must interact with, and each such system has languages, or APIs, that do precisely that. Modern databases created in the last one or two decades deviate from the traditional way of doing things, but relational databases all share the common trade that they're based on tables with precisely defined column names and types. On top of that, relational databases all come equipped with a common query language called structured query language (SQL). There are variants and dialects of SQL, but fundamentally it is an API used to interact with relational databases: It is used to create, alter and delete tables, insert entries or modify them, retrieve and filter data, and so on. The internal structure of (relational) databases is fascinatingly complex and a lot of optimizations carefully hidden from the user go into modern database design. In fact, there are a lot of different ways to implement a relational database, including many different data structures and algorithms for data storage and retrieval to choose from, but to the user this is all a black box. All data access happens exclusively through the domain-specific language SQL.

Remote procedure calls

So far, we've only explicitly discussed APIs that relate to work on a single process on a single computer. However, it's normal to have several processes running on a computer at practically all times. What's more is that these processes have to communicate quite often: For instance, if you're running a text processing tool to take notes and host a meeting with a conferencing software at the same time, your operating system is smart enough to copy a note from the former into the chat of the latter. That's a form of inter-process communication (IPC). Going one step further, your colleague who sits at another computer entirely can read your note in his instance of the same conferencing tool running as a process on his computer. Letting computers "talk" to each other like this is a complicated topic that usually involves complicated communication protocols. An essential part of communication between several computers is networking, i.e., the way the computers are connected with each other, for instance over the internet. One specific IPC paradigm between different machines, originating from the field of distributed computing, is called remote procedure call (RPC), which is a request-response protocol. It means there are two parties involved: a client and a server, connected to each other via some network. The client sends a request to the server to let it execute a program with parameters as specified. The server runs this program accordingly and returns the result to the caller, the client. The tricky part is that the program itself must be transferred over the network from client to server and the result must also be transferred from server to client. The complication stems from the fact that all in-memory objects of the client that are necessary for the function call have to be persisted first, sent over the network, and then read into memory again by the server. Many RPC systems use an interface description language (IDC) to describe a contract, which can then be used to generate code for both client and server. A prominent and widely adopted RPC system is Google's gRPC, which uses Google's Protocol Buffer format as IDC and the web-wide standard hypertext transfer protocol (HTTP) for network transport. HTTP is one of the core protocols on which the World Wide Web heavily relies and powers most modern web APIs.

Web interfaces

When you look at how rich and diverse modern web and mobile applications can be, it might strike you as surprising that almost all communication is backed by web protocols like HTTP. This protocol was officially introduced in 1997. HTTP/2 has existed since 2015, and its successor HTTP/3 is already supported by some browsers (Berners-Lee, 1996). The modern web is based on a set of constraints or principles called **representational state transfer (REST)**, and applications that utilize this set of principles are often referred to as RESTful applications or services. APIs for RESTful applications are referred to as REST APIs, and often HTTP is used for communication in such interfaces. The evolution of the HTTP protocol as an API is interesting in its own right, but we're going to focus solely on the semantics of HTTP and how you interact with it in practice. At its core, HTTP works with universal resource identifiers (URI), with which you can uniquely identify resources on the web, and operations or methods that tell HTTP what to do with the resource exactly. The basic process is that the client asks for an HTTP method to be executed, which the respective server corresponding to the URI does, after which the server then returns the result of the request to the client. The prototypical example is to type an address like `google.com` into your browser and hit enter. In this situation your browser is the client sending a GET request to Google's servers, who respond with the HTML code for the landing page of Google, which in turn can be rendered by your browser. The full list of basic HTTP methods goes as follows:

REST
The concept of representational state transfer (REST) was introduced in a PhD thesis in 2000 and specifies a set of architectural constraints for web applications, like client-server architecture or statelessness of services, which helped web application developers to find a common communicational ground to build the massive, distributed network that is the internet.

- GET requests to get a specified resource and it is only used for data retrieval. A GET request has no other side effects, on the server or otherwise.
- HEAD is identical to GET, with the exception that you don't receive any response body. For instance, when GET would give you the HTML code for a website, the corresponding HEAD request would only contain the meta information of the request (like information

about the server), but no actual content. A minimal web application must specify at least HEAD and GET methods to qualify as such, all the other following methods are optional.
- POST differs from a GET request in that it comes with a request body that is posted to and accepted by the server and used to run the request before returning to the client. An example would be posting to a message forum, in which case the text of your post is contained in the request body.
- PUT is somewhat related to POST in that it also sends content through the request body. However, it is semantically different because you require the server to store the body, i.e., put the sent resource there.
- DELETE, unsurprisingly, deletes a previously stored resource, which you might have PUT there.
- PATCH modifies a previously stored resource, if only partially.
- OPTIONS returns to the client the HTTP methods supported by the server.
- TRACE is essentially there for debugging purposes and echoes incoming requests back to the client for it to check whether the original request has been modified or amended by the server.
- CONNECT is method in which the client asks the server to be connected with another machine or through another protocol. In this situation the server acts as a proxy. This method is often used to set up a secure client-server connection through securely encrypted methods like **HTTPS**.

HTTPS
An extension of HTTP, HTTPS uses a cryptographic protocol-like transport layer security (TLS) or its predecessor secure socket layer (SSL) to secure the connection between clients and servers and to prevent fraudulent behavior on the internet.

Graphical user interfaces

Lastly, graphical user interfaces (GUI) can also serve as a form of API. The operating system you're using likely comes with a graphical interface and provides you with a graphical representation of the folder structure on your computer, a desktop, a mouse cursor, and other amenities. In the early days of computing all you had was a CLI, and in the very beginning not even that. Today GUIs are the default for most commercial applications running on desktop computers or smartphones. One particularly relevant application class of GUIs in the context of data-intensive sciences is that of integrated development environments (IDE), which give programmers a complete support system for their development efforts. Tools like PyCharm or Rodeo are popular choices for Python programmers and data scientists who want to be productive in their daily work. Such IDEs offer tooling for code completion, navigating complex projects, finding specific source code parts, debugging your code, or profiling the run-time performance of your programs. In a sense, IDEs provide you with an API for all things directly related to your programming tasks, and it is difficult to imagine such powerful tools without the graphical interface.

4.2 API Design Principles

You've now seen the many forms in which APIs can come and what they look like. It's time to investigate how to build APIs in practice and get a feeling for good and bad designs by understanding some core principles of API design. Simply put, when writing anything, and especially code, you want to be understood. To start out, let us have a look at two concrete examples of machine learning software workflows. The first one is an example using

Ray, an up-and-coming distributed computing framework for Python that supports a variety of algorithms for a machine learning paradigm called reinforcement learning. You install this Python library with `pip install ‘ray[rllib]’` and to check if the installation worked you could run

Code
```
import ray
ray.init()
```

in a Python session. That should result in no errors, so we should be able to run a first example by typing the following line in your shell:

Code
```
rllib train --run=PPO --env=CartPole-v0
```

When you do so, you will notice that this program does not run correctly; it gets interrupted by an error message embedded in an output that’s simply too long to put here in its entirety. After sifting through about 80 lines of code you eventually find a clue:

Code
```
ImportError: Could not import tensorflow
```

The reason is that this Ray example relies on Google’s TensorFlow library, which you’d have to install first. The problem with this example is not that Ray didn’t install TensorFlow, which in fact it shouldn’t. It’s that we verified our installation, but Ray didn’t complain; it happily accepted our setup and started running the example. What’s more, the error message is too difficult to read and novices have a hard time extracting information from such long messages. The last reason why this example is problematic is that the error message occurred too late in the program flow. The error occurs after Ray has set up its distributed computing framework, which is why you find the relevant error message about TensorFlow only after about 50 lines of program output. Now, the reason we’re discussing this example at great length is that it’s an example of bad API design. Imagine you run a complicated machine learning experiment with hundreds of lines of code. You start the experiment in the evening to see the results in the morning, only to notice that you forgot to install TensorFlow and the program crashed right away. That can mean a huge dip in productivity and it all comes down to a questionable design choice in error handling. In contrast, let’s see how a different machine learning framework handles the exact same situation. This time we’re looking at Keras, a popular library for deep neural networks. As before with Ray, it’s not necessary for you to know anything about machine learning, we’re just concerned with API design in this section. After installing Keras with `pip install keras`, you can validate the installation by typing

Code
```
import keras
```

into an interactive Python session. What you'll see right away is the following error message:

Code
```
ImportError: Keras requires TensorFlow 2.2 or higher. Install TensorFlow
via `pip install tensorflow`
```

Note how Keras not only tells you immediately that something went wrong, it also tells you what exactly it requires and suggests a concrete action to solve the problem in human-readable form without searching through a massive wall of text output. This type of error handling demonstrates excellent API design and goes to show that well-written APIs can save you a lot of problems down the road. A good API improves developer experience and enhances productivity. Keras has been hailed for its API design since its inception in 2015 and in recent years understands itself less as a concrete library and more as an API specification that you can implement and follow (Sadrach, 2020). Since the release of TensorFlow 2.0, previously a standalone library, Keras is now an officially supported API of TensorFlow (Chollet, 2017). We'll get back to Keras in this section to illustrate a few more points about good design.

The Zen of Python

One of the reasons the Python programming language is recognized as elegant and easy to start with is that it has been built with a design philosophy in mind that has been eternalized in the so called "Zen of Python." The Zen is a list of flexible principles, not to be confused with a rule set, which you can read using any Python version by typing `import this` into a Python session. We'll go through these principles in detail, as they make some good points about API design and its difficulties. On a side note, this philosophy has been lovingly mocked by Daniel Greenfield with an Anti-Zen library called `that`, which shows you anti-patterns to avoid. An aggregated view on the Zen of Python reads as follows:

- Beautiful is better than ugly. Python wants you to prefer beautiful solutions over ugly ones, which can be highly subjective. Working with intuitive concepts that click has merits for your users.
- Explicit is better than implicit. Hiding information from your users might require them to know about your implicit rules, so be as explicit as possible.
- Simple is better than complex, complex is better than complicated. The Python equivalent of Occam's razor tells you to shed complexity whenever possible. If you have to incur complexity, at least make sure not to complicate things.
- Flat is better than nested, sparse is better than dense. Prefer data structures and concepts that aren't too entangled and nested. For instance, a flat Python dictionary with properties is usually preferred to a nested one. Also, don't cram too many things into one line of code in order to avoid too much density.
- Readability counts. Needless to say, code that's easier to follow is more fun to read and easier to maintain long-term.

- Special cases aren't special enough to break the rules, although practicality beats purity. This is a very interesting part of the Zen, since it shows you on a meta level that you have to take these principles with a grain of salt. While you should not usually let special cases ruin your overall design, that is exactly what might be necessary sometimes (representing an exception to the rule).
- Errors should never pass silently, unless explicitly silenced. When errors occur fast and quickly, the user never has to guess what the reason for a program failure might be. Doing so might make your program flow more complicated but it is a good design choice. In rare cases, when it's safe to do so, error messages can be silenced.
- In the face of ambiguity, refuse the temptation to guess. In other words, try to avoid ambiguity by being as explicit and clear as possible.
- There should be one—and preferably only one—obvious way to do it, although that way may not be obvious at first unless you're Dutch: This reference on the Dutch creator of Python Guido Van Rossum is probably the most controversial of all the principles on this list. Python programmers refer to this way of doing things as "pythonic." Often there's no doubt about a pythonic solution and the community largely agrees on certain patterns, but sometimes it's just not clear if one approach is actually better than another. Still, the point is to strive for code that's so obvious to use that you can't go wrong.
- Now is better than never, although never is often better than right now. When designing things, don't delay your process and find a solution now, even if it is sub-optimal. You can iterate and improve on it later. However, if hard-pressed, don't go down a road that makes things too difficult to fix later on.
- If the implementation is hard to explain, it's a bad idea; if the implementation is easy to explain, it may be a good idea. This slightly pessimistic view on programming yet again alludes to beauty and understandability. Not everything that's easy to explain is good, but if you can't explain it, it's certainly bad.
- Namespaces are one honking great idea—let's do more of those! Python makes ample use of namespaces to modularize and structure its code. Let's say you implement a `zoo` with plenty of animals in Python. Using proper name-spacing, your `Elephant` class might be imported as `from zoo.animals.mammals import Elephant`, instead of `from zoo import Elephant`, as the latter might clutter your namespace with too many concepts.

Separating Interfaces and Implementations

A common principle in interface design is to separate interfaces and their implementations. Strictly speaking, by the definition we've given and the properties discussed, an API is simply an interface, and the whole point is that its users don't have to care about implementation details. In practice, however, it's good to be reminded of this principle sometimes. Interface separation can be broadly understood within the principle of **separation of concerns**. Strongly typed languages like Java or Scala usually have a dedicated class for interfaces, which enforce this practice. In Java, an "interface" can't have a concrete implementation, as it is prohibited by design. If you want to work with functionality as laid out in the interface, you have to provide an implementation of the interface by writing a subclass for it. On a more fundamental level, the two programming languages Java and Scala, which are syntactically quite different, compile to compatible byte code on the Java virtual machine (JVM). In particular, you can import Java classes into your Scala code and build software on top of it. In Python, there is no such concept as an interface in the basic language, and a strict separation of interfaces and concrete implementations are considered an anti-pattern (or un-pythonic). However, that does not completely invalidate the principle just presented. In Python, you still want to carefully think about the objects your users will interact with, the public API of your code, and the rest of the code, the implementation. If you view the public-facing API as a "surface area" which most of your users will interact with, for larger projects this is really just the tip of the iceberg, which makes good interface design even more important.

Separation of concerns
In software engineering, a common principle is to separate concerns, which means that you build your software in such a way that each part or module is solely responsible for certain aspects and concerns should never be divided by two or more such modules.

Abstractions

When building APIs, an important aspect is to find the right abstractions, i.e., the right concepts that go well with what you intend to build. Abstractions should feel natural and make it easy for your users to interact with them. As famous computer scientist Edsger Dijkstra puts it, "[b]eing abstract is something profoundly different from being vague... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise" (Misa & Phillip, 2010, p. 1). To give you one example, the Keras library gets away with very few high-level concepts. To illustrate the elegance of Keras' approach you need to know that deep neural networks, which Keras is built for, are machine learning models built from layers of relatively simple compute instructions. The simplest models are sequential in nature, which means that the layers involved follow one after another. The simplest layer in a neural network is probably a densely connected layer, meaning that all neurons (a term borrowed from neuroscience used to refer to how the brain works) in one layer are connected to that of the next. Even if you've never heard about deep neural networks before, now that you're equipped with this knowledge, you can read the following Keras model specification:

Code
```
import keras
from keras import layers
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

In other words, to construct a sequential network with Keras, you sequentially add as many layers as you would like to. The specifics of this model are irrelevant here, just note that it almost reads like plain English and works with precisely the same high-level abstraction we just explained: models and layers. You might say that this is an obvious thing to do, but far from it: Before Keras, none of the deep learning frameworks adopted a similarly simple and intuitive style of programming, which means the discipline of deep learning was much less accessible to practitioners. In 2020, the landscape has changed for the better, but the impact of well-written APIs such as Keras on the hype of deep learning and machine learning in general is not to be underestimated. It is famously said that all abstractions are leaky (Spolsky, 2002), meaning that no matter how well you think through an abstraction in software engineering, there will be cases at which you will have to look "under the covers" or break the rules for special cases. This reflects the fact that the world can be messy, and abstractions are just an attempt at grouping things into categories, a process which can fail for several reasons. While that statement is true, it is also often irrelevant in practice to the extent that good abstractions go a long way, while poor choices might make you fail badly. Another way to reason about abstractions, especially when it comes to leakiness, is that you might require different levels of abstraction. Sometimes you're doing just fine with the highest level of abstractions, but sometimes you have to go a few levels deeper to adjust code to your use case.

Error Handling

We've already seen good and bad error handling in an earlier comparison of the Ray and Keras Python libraries, but it's worth generalizing and summarizing a few points here that are important to keep in mind:

- When throwing an error, fail as early and quickly as possible. You do this for maximum transparency and to prevent more serious errors down the line, which might be more difficult to understand and fix.
- Your error messages should be human-readable and clear. They should not only give the user a local error message that is hard to interpret, but they should also give the context in which the error occurred and what the user can do to fix it.
- Provide users with messages at different levels, for instance, provide useful warnings about a deprecated function that might be removed from the next release.

Convention and Configuration

According to computer scientist Phil Karlton, "There are only two hard things in Computer Science: cache invalidation and naming things" (Fowler, 2009, p. 1). While this is a bit tongue-in-cheek, there's a lot of truth to that statement. When it comes to API design, there's mostly just one part of the quote that is relevant: naming things. Finding good naming conventions is surprisingly hard and some languages, like Java, have a cultural tendency to have very long class names. Take `AbstractTransactionalDataSourceSpringContextTests` from the Spring framework, for example. That is a great example in the sense that it should be avoided, and likely points to either a conceptual lack of clarity in your abstraction or the fact that your class simply does too much (and hence violates the separation of concerns principle). Giving clear and precise names to your classes and functions should be something you always keep in mind when designing software. Apart from naming things, other types of conventions are important, too. For instance, in Python you can have default arguments in your function signatures. You should always make good use of them, whenever you have a sensible default to put there. Most of your users won't be power users and don't care for every single parameter that you can tweak. Having said that, it's better to have a lot of parameters in your function parameters that your users can configure, than it is to let them modify them through code. To give a concrete Python example, let's say we have a simple class encapsulating a bucket, which has a size and can potentially be filled with something. Here's a Python class for that:

Code
```
class Bucket:
def __init__(self, size=10):
       self.size = size

def update_size(self, size):
       self.size = size
```

The default bucket size is 10 (the meaning or unit of this number is unimportant here), so users can create a bucket using this convention with

Code
```
bucket = Bucket()
```

They can also configure the bucket size upon initialization using

Code
```
bucket = Bucket(size=20)
```

And finally, users could also first create a bucket and then modify its size through code (a questionable operation altogether):

Code
```
bucket = Bucket()
bucket.set_size(20)
```

A good rule of thumb to follow is "convention over configuration over code." Provide as many good default values as possible and make them easily configurable.

User Experience

A lot of the principles discussed in this section come down to applying common sense reasoning to the domain of software engineering. Ultimately that's because building good APIs, or designing things in general, is all about empathizing with your users and respecting the cognitive factors that come into play when building things for humans. In other words, user experience, also called developer experience when applied specifically to the relationship of engineers to your code, is important. The cognitive dimensions framework (Clarke, 2004) lists a total of 12 points in relation to how software engineers work with APIs and what they expect from them.

- Abstraction level: What's the maximum and the minimum number of abstractions found in your API, and what comes in-between? What level should your users work with primarily?
- Learning style: What are the requirements for learning the API? Does it have a steep learning curve? What's the target group of developers, and what's their background?

- Working framework: What's the cognitive load or conceptual chunk needed to understand this API and work with it effectively?
- Work-step unit: How much work is required by the user in each step? Is the API particularly dense in that regard?
- Progressive evaluation: Can you build up code parts progressively and get feedback on your progress? Or do you need, in extreme cases, to bring all pieces together to see if an idea works?
- Premature commitment: If I go down one path, how many other non-trivial follow-up decisions do I have to keep in mind as a consequence of this decision?
- Penetrability: How well can you explore the API on your own? Is it well documented and can you understand and compose its components intuitively?
- API elaboration: How tailored to your needs is the API? Are there possibilities to flexibly configure it, or do you have to modify it heavily to suit your needs?
- API viscosity: Is it easy to modify or extend the API for a different use case? Does it speak the language of the developer community it targets?
- Consistency: If you've seen one part of the API, can you reason about the workings of a different yet related part of it? In other words, is the API consistent in its choice of building blocks?
- Role expressiveness: Do the abstractions picked by the API match your cognitive model of the components as they relate to the program flow overall?
- Domain correspondence: All APIs are written within a context, for a specific domain. How well does the API at hand express the models present in that domain?

Documentation

Lastly, good documentation is crucial for any successful API with a large user base. Make it a habit to

- document your code.
- document your interfaces, input parameters, and return values.
- document the intended use of your API.

4.3 Building a Python Library

Having looked at various API paradigms and design principles, it's now time to build a concrete Python library to apply what you've learned. The focus of this library is not so much about the implementation details of the library as it is about its API. In fact, the code we're writing behind the API will be relatively complicated in order to show you how good API design can hide it. To offer a concrete example, let's say you run a company specializing in analyzing large corpuses of text documents, and you sit on a proprietary algorithm that can do large-scale analyses faster than everyone else. Of course, you want to bring that algorithm in the form of a service to your users, to provide value and make money. The first step for you is to write a Python library that properly encapsulates your algorithm and makes it accessible for users, without sharing your secret algorithm. To restrict things even further, we only do one thing: count all occurrences of words across several documents.

An Algorithm to Summarize Documents

Let's start with the implementation of the core algorithm first. This is a toy version of a programming paradigm called MapReduce, which is a milestone in distributed computing and one of the many impressive contributions to large-scale computing by Google (Dean & Ghemawat, 2004). Many commercially successful big data technologies like Hadoop are based on this programming model. It is roughly based on three steps:

1. Take a set of documents and transform or "map" the elements you're interested in (for instance, the words of the document) according to a function you provide. This step returns key-value pairs, in which the keys are the document elements you care about and the value is an entity you want to compute. In our use case we're interested in counting words, so each word `x` in a document would result in the key-value pair (`x, 1`), where the "1" signifies that we counted the word once.
2. Collect and group all the output pairs from the last step. Let's say the word `x` appeared four times throughout the documents. The grouping step would then result in the value `x: [1, 1, 1, 1]`.
3. The last step is to aggregate or "reduce" the elements from the last stage. In our case, we just want to sum all values for a total count. In the example of the word `x`, we would therefore get the result `x: 4`.

As you might guess, the paradigm MapReduce gets its name from stages 1 and 3, but the second one is important as well. While the three steps may look simple, their power lies in the fact that they can be massively parallelized across hundreds of machines. Let's provide a naïve implementation of this algorithm for our use case, word counting, in Python. We start with implementations of the three phases. First, we define a map function that returns (word, 1) as key-value pair for each word in the text:

Code
```
def map_function(text):
   for word in text.lower().split():
   yield word, 1
```

Then, we apply the map function to all elements in our data set `data` (a list of text data):

Code
```
def apply_map(data):
   map_results = []
   for element in data:
       for map_result in map_function(element):
           map_results.append(map_result)
   return map_results
```

To group the key-value pairs produced by the map step by their key, we do the following:

Code
```
def group_function(map_results):
   group_results = dict()
   for key, value in map_results:
       if key not in group_results:
           group_results[key] = []
           group_results[key].append(value)
   return group_results
```

Finally, we define a reduce function that sums up the counts of words:

Code
```
def reduce_function(key, values):
   total = 0
   for count in values:
       total += count
   return key, total
```

As with the map function, we also need to apply the reduce function to all key-value pairs from the group phase:

Code
```
def apply_reduce(group_results):
   reduce_results = dict()
   for key, values in group_results.items():
       _, count = reduce_function(key, values)
       reduce_results[key] = count
   return reduce_results
```

To see how this all works, let’s create a test data set to run this program:

Code
```
text_1 = "Lorem ipsum dolor sit amet, consetetur et sadipscing elitr."
text_1b = "Lorem ipsum dolor sit amet, consetetur et sadipscing elitr."
text_2 = "At vero lorem et accusam et justo duo ipsum et ea rebum."
text_2b = "At vero lorem et accusam et justo duo ipsum et ea rebum. At vero lorem et
accusam et justo duo ipsum et ea rebum."
data_set = [text_1, text_1b, text_2, text_2b]
```

When we now apply all three MapReduce stages sequentially, we get the following output:

Code
```
map_results = apply_map(data_set) # 1. stage
print(map_results) # Output: [('lorem', 1), ('ipsum', 1), ('dolor', 1), ...]
group_results = group_function(map_results) # 2. stage
print(group_results) # Output: {'lorem': [1, 1, 1, 1, 1], 'ipsum':
```

```
[1, 1, 1, 1, 1], 'dolor': [1, 1] ...}
reduce_results = apply_reduce(group_results) # 3. stage
print(reduce_results) # Output: {'lorem': 5, 'ipsum': 5, 'dolor': 2 ...}
```

Now that we've checked that our code works as intended, let's take a step back. We've implemented a toy version of a powerful, commercially successful algorithm and applied it to a concrete use case. However, we do not have an API for this algorithm yet. The last thing we want to do is share our intellectual property, so we need to carefully hide all implementation details and provide users with a simple interface to analyze their documents. How can we do that? First, we have to clarify the specifications of the API by understanding what our users might want from us. Let's say we create a minimum viable product (MVP) by constructing the following Python functionality that we can share with users:

Code
```
def count_words_naive(corpus):
   map_results = apply_map(data_set)
   group_results = group_function(map_results)
   reduce_results = apply_reduce(group_results)
   return reduce_results
```

Of course, to protect our IP, users will only have access to `count_words_naive`, not its implementation. After an initial round of interviews with potential customers we get the following feedback which we can use to refine our approach:

- Many users didn't understand how to use this function. The input argument `corpus` is not documented and it's not clear how to use it.
- Some users figured out that our API takes a list of Python strings, representing the text documents to analyze, but had the problem that the function sometimes throws obscure errors, and they didn't know why.
- Users who got the result sometimes wanted something slightly different: They were interested in words with a minimum number of occurrences in the corpus (otherwise the output is too large).
- Some users were not proficient enough in Python and just wanted to provide a list of files, instead of loading documents into Python themselves first.

To incorporate this feedback, we first design a new backend function to filter words whose count is too low, which looks as follows:

Code

```
def filter_function(results, min_occurrences=4):
   return {k: v for k, v in results.items() \
           if v >= min_occurrences}
```

With this filter function, we can now create a much-improved interface by taking several design principles from earlier into account, namely, proper and intuitive error handling, good documentation, using good conventions and configurations, and even making use of Python's type checking module `typing` for input parameters and return types:

Code

```
from typing import List, Optional, Dict
import warnings

def count_words(corpus: List[str], filter: Optional[int] = None) -> Dict[str, int]:
   """Count all words in a corpus of documents.
   :param corpus: A list of strings, where each string
                  contains the text of a document.
   :param filter: int or None. If None, don't filter
                  the result. Else return words
                  with at least 'filter' occurrences
                  in the corpus.
   :return: A dictionary of words and their respective
            counts, filtered by count as specified.
   """
   assert type(corpus) is list, "The corpus to analyse needs to be list of strings"
   map_results = apply_map(corpus)
   group_results = group_function(map_results)
   reduce_results = apply_reduce(group_results)
   if filter is None:
       return reduce_results
   else:
       assert type(filter) is int, f"The 'filter' argument needs to be a Python int,
       you specified: '{filter}' which is of type {type(filter)}"
       return filter_function(reduce_results, min_occurrences=filter)
```

Note that, while this might look like much more than we had in our naïve definition, most of the additional lines of code go into type checking, documentation, and catching errors. That is a good sign and mature interfaces tend to look this way. To address the need for an API call that works with files instead of in-memory strings, we can define a new interface that calls the above `count_words` function internally. Reusing function calls instead of creating separate code bases is another good design choice.

Code

```
def count_words_from_files(files, skip_corrupted_files=False, filter=None):
   """Count all words in a corpus of documents provided        as files
   :param files: A list of file names (str) on your local file system. The files cont
    the text you want to count words for.
   :param skip_corrupted_files: boolean. If True, ignore all files that can't be reac
    otherwise abort.
   ...
```

```
    """
    corpus = []
    for file_name in files:
        with open(file_name, 'r') as f:
            try:
                text = f.read()
                corpus.append(text)
            except:
                msg = f"The file {file_name} cannot be read. Remove it from the corpus."
                    if skip_corrupted_files:
                        warnings.warn(msg)
                    else:
                        raise ValueError(msg)
    return count_words(corpus, filter) # reuse function
```

There is a lot to be unpacked in this piece of code. First, the arguments of `count_words_from_files` are different from `count_words` in many ways. The "files" parameter still takes a list of Python strings, but the meaning is different now. As the documentation explains, "files" is now a list of paths on your local system to files that should be analyzed. The "filter" parameter is the same as before, but we have a new parameter: `skip_corrupted_files`. This parameter accounts for the fact that not every file on your system can be opened and contains text data. `skip_corrupted_files` is false by default, which means that the program will stop immediately when it finds a file that can't be read. This is a smart default, as it's more secure than just skipping such an event, but users might want to turn this feature off at their own risk. Hence, we expose this parameter to allow users to configure it as they need. As for the code itself, the flow is straightforward. We iterate over all files and try to read them, and if we succeed, we pass the result into the `count_words` function defined before.

SUMMARY

In this unit you've learned the fundamentals of application programming interfaces (APIs), their design, and their implementation to get you started with data-intensive science projects on that subject. All programming examples in this unit were given in Python, as it is easy to prototype with and currently the most relevant language when it comes to data science and related fields. The first section exposed you to the basic terminology of APIs and how various levels of abstraction, from operating systems and hardware access to modern web interfaces, come into play when working productively with a computer. These layers of APIs span a wide class of important entry points for application development and include database access, programming languages, and the use of your preferred toolchain in data-intensive fields. You now understand APIs as specifications between the API designer and user, a binding contract that parties must adhere to. The focus of this section was on the usage of libraries and frameworks, as well as the most important

building blocks of RESTful web applications. Next, we had a closer look at what it means to actively build APIs, by introducing key design principles, dos and don'ts. You've learned about the importance of separation of concerns, good error handling, sufficient documentation, finding good abstractions that aren't too leaky, and using conventions and configurations to help users navigate your APIs in practice. The following section gave you a concrete use case of a small Python library built to share with your users. We built out a relatively complex underlying algorithm to count words in documents and focused on building a minimal user-facing API to expose it. You've seen how to effectively hide functionality that's not necessary for your users to know, document your code unambiguously, use type hints to improve validation of input and output arguments, and handle errors properly in order to catch unintended behavior early on. To conclude, you can now identify strengths and weaknesses of APIs when you're working with them, build them effectively, and set up your own libraries and REST interfaces in Python.

UNIT 5

FROM MODEL TO PRODUCTION

STUDY GOALS

On completion of this unit, you will have learned …

- the details of the process of developing a machine learning model, and how it differs from the standard process of software development.
- the lifecycle of a model once it reaches production, including updating, monitoring, health checks, performance, and scaling.
- the specifics of deploying a model and keeping it deployed.
- the purpose of MLOps and DataOps.
- how end-to-end cloud services offer complete solutions to all the issues above, specifically covering AWS SageMaker.

5. FROM MODEL TO PRODUCTION

Introduction

A machine learning solution provides predictions based on a set of input data in a non-deterministic way. Its development is an iterative and explorative process. We begin by identifying and clarifying a case where machine learning can bring value to an organization. We then proceed to collect and explore the available data, generate a hypothesis on the nature of the data, the problem, and the solution, that we then explore and test both visually and programmatically in machine learning models. Once we have a solution candidate, we proceed to integrate it in the existing infrastructure, or develop software to take advantage of it. We will explore how the machine learning development process has similarities and differences with the traditional software development process. Bringing a machine learning solution to production presents specific challenges due to its non-deterministic nature, the nature of the teams developing it, and its intrinsic dependence on the data used to train it, as well as the data it is used on. We will explore the specific challenges of version control, testing, and monitoring due to the nature of the machine learning solution. Later, we will begin to see how some tools offer support for several of the challenges of production machine learning, introducing the model of MLOps, and how continuous integration (CI) and continuous delivery (CD) models of DevOps can be applied and extended to machine learning solutions, with the addition of continuous training (CT). We will speak in more depth on the existing solutions with MLFlow, Kubeflow, and Michelangelo. We will then explore how existing cloud providers offer solutions to the challenges we are facing, going into more specific detail in the case of Amazon SageMaker.

5.1 Model Development Lifecycle

In this unit we are concentrating on the challenges of bringing a model to production, so it is useful to start from an overview of the lifecycle of model development. Machine learning involves creating a model trained on certain training data which can afterward process additional data to make predictions. In this context, a **machine learning model**, or a statistical model, can be considered a black box that, given a set of inputs, returns some outputs. The basic idea is that the model allows us to map a set of inputs to a desired output. For example, we want to train a computer to classify an image as a cat or a dog. In this example, we give the model input images and labels telling the model whether the image is a cat or a dog. The black box then learns an association between inputs and labels, so that it can hopefully predict the right answer (cat or dog) for previously unseen images.

Machine learning model
This is a file that has been trained to recognize certain types of patterns.

To develop a model that brings value to an organization, the basic steps are

1. understanding the problem,
2. finding good assumptions and hypotheses,
3. collecting the available data,
4. exploring it to see if the assumptions and hypothesis seem realistic,

5. experimenting with possible models and data processing steps to solve our problem,
6. training a model, and finally
7. deploying it on a development pipeline to make sure it behaves as expected.

Let's delve deeper, and please note that all steps are connected to each other and can't be seen in isolation. As an example, while cleaning the data, we also analyze them a bit, and after exploring, we sometimes need to go back and change our cleaning process, because it removed too much information together with the noise, or because it didn't remove enough.

Problem Understanding and Hypotheses

First, it is essential to understand the problem you're dealing with. In the best of cases we are given a very narrowly defined project scope. However, requirements are often vague, like "improving turnover," "reducing churn," or "identifying customers that won't pay." Finding the right answers to what is actually needed to succeed with your project is called requirement engineering. Various project management techniques need to be carefully applied to achieve this. The process of understanding and defining the problem can involve long interviews with matter experts, design thinking, and exploratory modeling. We often have to go back and forth checking and correcting our initial hypotheses and assumptions. The field of data science is still comparatively new and many companies struggle to adopt a data-driven mindset, meaning their stakeholders have a hard time setting up data science projects for success. For this reason, it is important to have a clear idea of what we want to solve and have clear performance indicators. This can be done by establishing well-defined, specific problems that can be answered. Rather than simply "identifying problematic customers," a goal such as "we want to reduce the number of customers not paying after 2 weeks by at least 20 percent" is much more concrete. As another example, even something that sounds more specific, such as "identifying the vehicles entering a building" could be improved upon to set the goal of "at least 95% of the vehicles entering a building are correctly identified." Some problems can be very hard to define, as in the case of machine translation: How do we score translations from best to worst? The answer to that is highly subjective, and while we somehow know, as humans, when a translation is really bad, it is difficult to evaluate good ones. Should they be very close to the original, should they keep the essence and take liberties, or something in the middle? Once we have defined the problem, we can start making hypotheses about what could solve it, what we could need to accomplish it, what assumptions we need to check, what correlations to have for mapping inputs to outputs, and begin researching different possibilities to accomplish that.

Data Collection, Cleaning, and Wrangling

Once we understand the problem and have a hypothesis for how to tackle it, we have to collect the data to work with. In some cases, collecting data can be as simple as getting access to a database with all the data tagged in the way we need. In other cases, it is much more complicated and can often involve a process of **ETL** (extract, transform, load). In some other cases, we can realize the organization does not have the data we need, but they can be accessed from a public source. There are several open data repositories, with

ETL (extract, transform, load)
This is a type of data integration that refers to the three steps used to blend data from multiple sources.

more being added every day. The problems with open data are that, since they are open, they generally won't offer much of a competitive advantage and, above all, they rarely meet our exact needs.

Another possibility is collecting the data ourselves. This is generally more time and resource intensive, but it allows us to obtain exactly what we need to solve the problem, or at least the best we can get. Examples include

- collecting movement data from passengers,
- taking photos from the camera we plan to ship with a product,
- equipping sensors on devices to monitor their behavior, or
- getting humans to tag pieces of existing data for supervised learning.

Data scraping
This is a technique in which a computer program extracts data from human-readable output coming from another program.

How to properly collect the data we need is both an art and science in itself, often overlapping with experiment planning. Another, often simpler possibility is to do **data scraping**, i.e., retrieving data from pages on the web or from sources not aimed at being directly consumed by computers, and transforming the data into computer-useable formats. It can involve getting data from websites, PDFs, or scanned documents. Scraping is time intensive, since it requires the creation of specific software for each individual source, and brittle since it is very tightly coupled to the source. Even a small change in the source will break the process and require it to be adapted. Additionally, scraped data can present legal problems that need to be evaluated if the data was not originally collected for the intended purpose (Import.io, 2017). Any sort of data (scraped, collected, or accessed from a database or datastore) will need cleaning and re-formatting for the intended purpose. This is another extensive topic, known as data wrangling or data munging. Data wrangling, or munging, is the process of transforming and mapping data from one data format into another to make the data more appropriate and valuable for our purpose. It involves transformations typically applied to distinct entities (fields, rows, columns, data values, etc.) within a data set, and could include actions such as extractions, parsing, joining, standardizing, augmenting, cleansing, consolidating, and filtering to create desired outputs (Wikipedia, 2020c).

Exploratory data analysis

Exploratory data analysis (EDA)
This is an approach to analyzing data sets to summarize their main characteristics, often with visual methods.

Once we have access to data, we can start working with them to get an idea of what they can tell us. This step is generally called **exploratory data analysis (EDA)**, and it refers to the process of performing initial investigations on data to discover patterns, spot anomalies, test hypothesis, and check assumptions, using summary statistics and graphical representations.

Standard tools for EDA help us to summarize the type and number of (distinct) values, type of data points, and median and mean of the data; create visualizations for an intuitive idea of what the data look like; and see what patterns can be extracted. Since humans are very visual, it is much easier for us to see patterns on an image than in thousands, or millions, of lines of data. EDA is an exploratory process; a data scientist will explore different possibilities, and delve deeper when something looks interesting. It is an intuitive process that requires experience, so we can only give a simple example of it. We will perform a bit of exploration of the iris dataset, a small dataset containing three classes of 50 instances each, where each class refers to a type of iris plant and is often used for demonstration purposes (Fisher, 1936). Typical first steps in EDA are checking the look of the data by inspecting the first few rows, which we do with pandas in Python (Pandas, n.d.). Note that we downloaded the iris dataset first in CSV format and stored it locally as a file called `iris.data.csv`. This is what the data look like in tabular form:

Code

```
import pandas as pd
iris_df = pd.read_csv('iris.data.csv')
iris_df.head() # first 5 rows
```

Table 3: First Lines of a Dataset with .head()

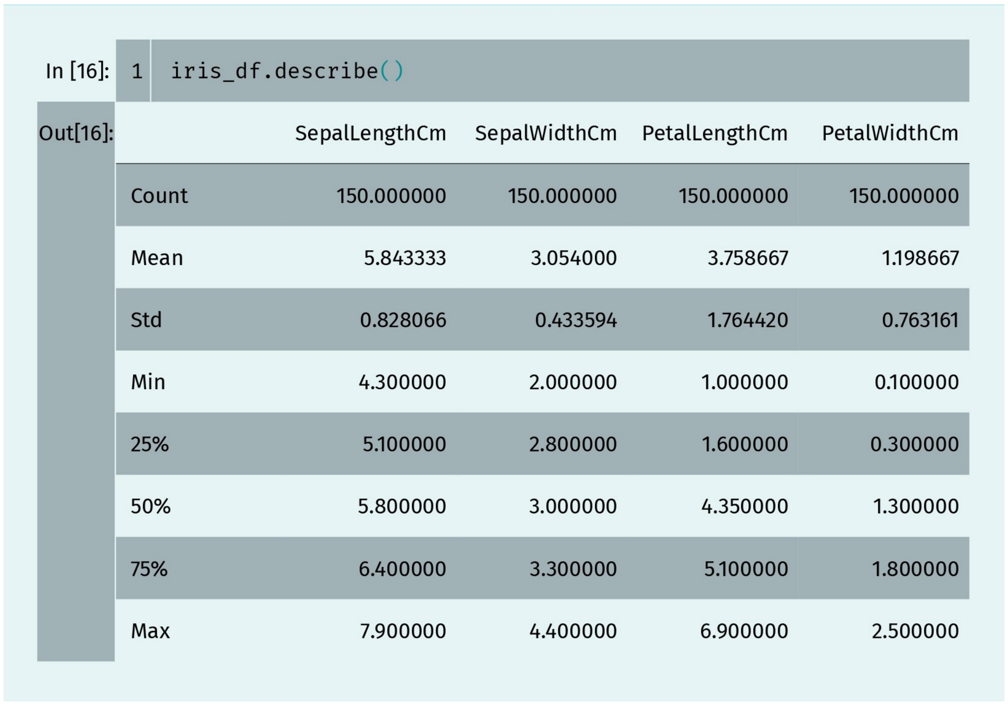
Out[15]:

	Sepal-LengthCm	SepalWidthCm	Petal-LengthCm	Petal-WidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	**4.6**	**3.1**	**1.5**	**0.2**	**Iris-setosa**
4	5.0	3.6	1.4	0.2	Iris-setosa

Source: Pedori (2020).

We see the dataset has four numeric values, and one text label called “Species.” We can compute summary statistics for the numeric values as follows:

Figure 47: Summary Statistics of a Dataset With .describe()


In [16]:

```
iris_df.describe()
```

Out[16]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Count	150.000000	150.000000	150.000000	150.000000
Mean	5.843333	3.054000	3.758667	1.198667
Std	0.828066	0.433594	1.764420	0.763161
Min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
Max	7.900000	4.400000	6.900000	2.500000

Source: Pedori (2020).

We can confirm we have a total of 150 values and get an idea of the average and distribution of values.

Experimentation and Feature and Model Selection

Once we have the data loaded in a useable way and we have an idea of what the data can tell us, we can start experimenting with models. We will test several different models, starting from simple models like linear or logistic regression, to more complex, like neural networks and XGBoost, depending on the use case. In the experimentation cycle, we will start with a few models and initially test them to get an idea of how they perform. Part of the process is selecting possible **features**. The data we have can contain meaningless noise, and some information could be extracted given our knowledge of the target domain. For example, if our data are about traffic, the day of the week is an important factor, but if our dataset contains only the day of the month, we will have to calculate the day and add it to the features. On the other hand, training a model incurs the curse of dimensionality. The more dimensions we have, the harder it is for a model to converge, so we sometimes want to perform **dimensionality reduction**. At this point we also usually begin considering possible models. The main type of machine learning problems are supervised, unsupervised, and reinforcement learning.

Feature selection
This consists of selecting a subset of relevant features to be used in the model.

Dimensionality reduction
This is the transformation of data from a high-dimensional space into a low-dimensional space while still keeping some meaningful properties of the original data.

- Supervised learning maps input data to known output data. Example: categorizing images as cats and dogs, or predicting the stock market.

- Unsupervised learning explores patterns in your data. Example: clustering documents by topics by looking at similarities in those documents, without specifying the topics beforehand.
- Reinforcement learning studies how agents interact with their environment by rewarding favorable situations and punishing bad ones. Example: learning to play a video or board game.

Choosing the right machine learning model for our task isn't easy. There are many machine learning models, and several Python software packages offer access to some, or many, of them. We tend to differentiate between traditional machine learning algorithms, mostly based on statistical methods, and deep learning algorithms, based on neural networks. In many cases, we get even better results with **ensemble models** that aggregate the results from several models to improve accuracy. It is sometimes the case that a first layer of machine learning is done by deep neural networks, and their results are then aggregated by simpler traditional methods. Once we have selected what model to try, we divide our data into **train, test, and validation datasets**, often with techniques like k-fold to maximize the amount of data we can use to train and validate.

Ensemble models
These use multiple models to obtain better predictive performance than could be obtained from any of the constituent models alone.

Train, test, and validation datasets
The test dataset is used for the initial fitting of the model, the validation dataset to evaluate the performance of the model at different steps, and the test dataset is used for a final unbiased evaluation of the performance.

Hyperparameter
This is a parameter whose value is used to control the learning process.

Hyperparameter Tuning

Every machine learning model has a set of parameters that are automatically adjusted during the learning process, and which we don't manually tweak. However, there are other, model-specific parameters called **hyperparameters** that can be chosen by the users of an algorithm. For example, in the case of deep neural networks you can specify the depth of a network. The way in which you select your features that will be fed into the model can also be seen as a hyperparameter of your machine learning experiment. Deciding on which hyperparameters to use is not easy, as the training process usually takes a lot of time, so you can't compute all possible combinations. Luckily, there are algorithms and libraries that help you with hyperparameter tuning. Using such a tool, the result will be hyperparameters optimized for the current dataset and application.

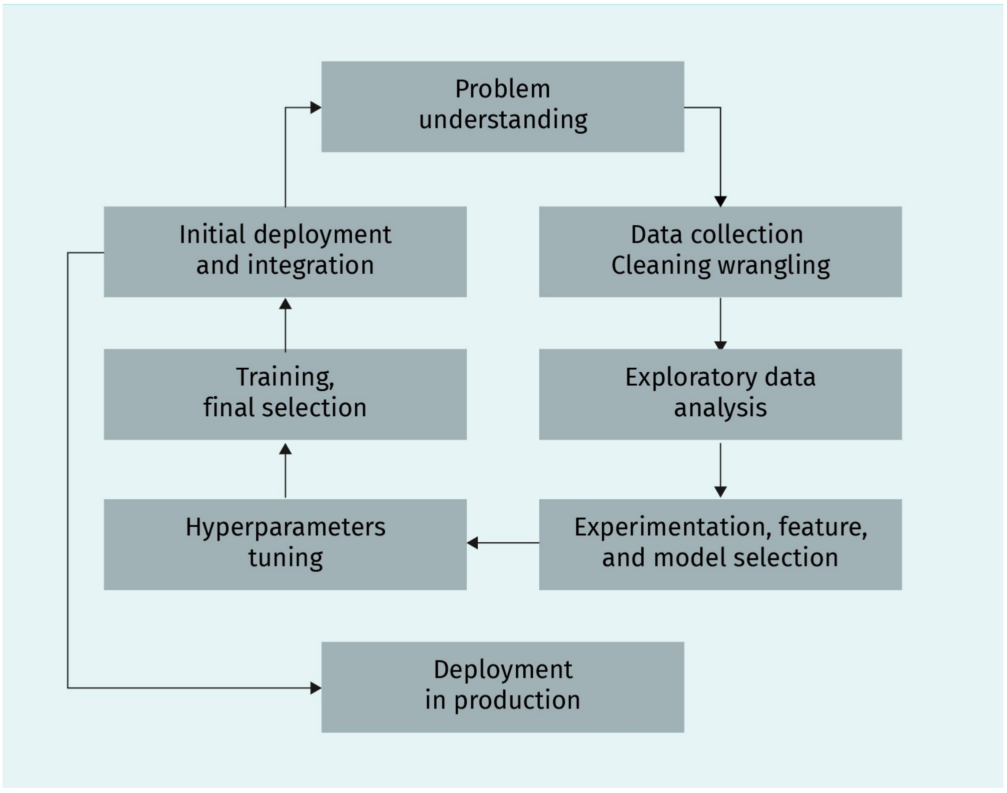
Training

We can now start training different models to compare their performance. We will usually train several models on the dataset for different choices of features. If our dataset is very large and training takes a lot of time, it is often useful to use only a subset of the dataset initially. Once you're confident in your choices (preprocessing, model, and feature selection), you can train your model again on the full data set. Depending on the amount of data and the use case, this can take quite a while, and lots of resources. To give an extreme example, GPT-3, by OpenAI, has 175 billion parameters and would require 355 years and $4,600,000 to train on current hardware (Brown et al., 2020). Our cases will not be as extreme, but we still have to do a complete training on the models we think are good candidates. This of course does not apply for models that can train in less than a few hours. At the end of the process, we have our trained models, and we can check their performance using the test dataset.

Initial Deployment and Integration

Once the model is ready, we need to make it available somehow. The easiest and most common way is to wrap it in a REST interface and deploy it as a microservice in a container. Once this is done, we can do some basic tests of usability, connect it to other services, and set it up in a development pipeline to make sure it behaves as expected.

Figure 48: Model Development and Production Lifecycle (1)


Problem understanding
Data collection Cleaning wrangling
Exploratory data analysis
Experimentation, feature, and model selection
Hyperparameters tuning
Training, final selection
Initial deployment and integration
Deployment in production

Source: Pedori (2020).

Differences to Software Engineering

Let's compare the traditional software engineering lifecycle to the machine learning lifecycle. In both cases it is necessary to understand the problem to be able to solve it correctly. However, in the traditional software world, the problem is often better defined, at least at the level of pure implementation. Engineers use quite common building blocks, and can iterate and check their assumptions. In traditional software engineering we talk less about hypotheses and more about requirements. It is easier to implement techniques for requirements-based testing as the results are less "fuzzy." The code is supposed to be deterministic, unlike machine learning models, even if some interactions can add complexity.

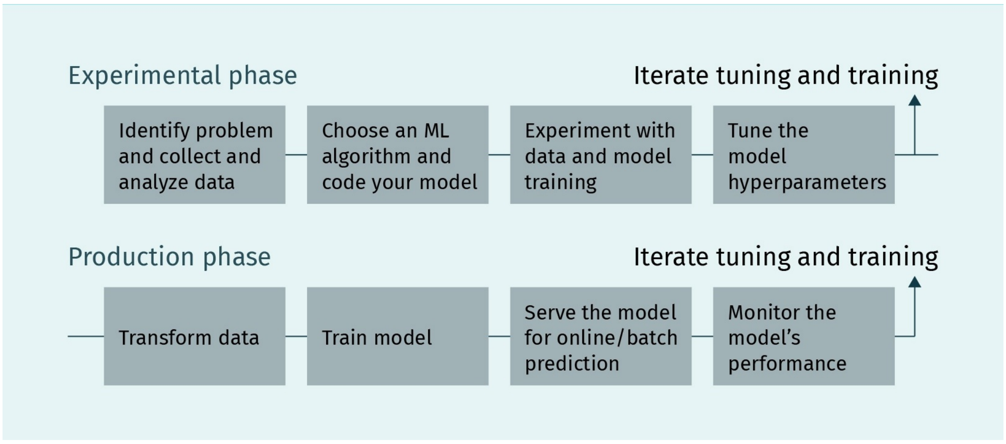
5.2 Model Production Lifecycle

Once our models are ready for production, the production requirements have many similarities to the production lifecycle of traditional software, as well as some specific differences. As with all code, it should be integrated in a continuous development and deployment pipeline, allowing for quick turnarounds and frequent shipping of reliable, secure, and available software. Using machine learning models to solve problems has been called **Software 2.0** (Karpathy, 2017). The complete ML system contains: the code to create the model, the trained model itself and other possible **artifacts**, and the data used to create the model. This code must be integrated in the pipeline described above. To bring and keep models in production, we then have to take care of versioning the code and storing and deploying all of the code. Additionally, we take care of the endpoints using our models, persist our predictions, log the results, and monitor health, performance, and what happens when our models are used on new or unseen data.

Software 2.0
This uses machine learning, specifically neural networks, to create a fundamental shift in how software is written.

Artifacts
An artifact is the output created by the training process.

Figure 49: Model Development and Production Lifecyle (I!)


Experimental phase
Iterate tuning and training
Identify problem and collect and analyze data
Choose an ML algorithm and code your model
Experiment with data and model training
Tune the model hyperparameters
Production phase
Iterate tuning and training
Transform data
Train model
Serve the model for online/batch prediction
Monitor the model's performance

Source: Pedori (2020).

Models in Production

When models reach production, in addition to all the usual challenges of production software, they encounter some specific ones due to the fact that they are based on software. We can think about some of the services most of us use practically every day.

Online search engines are, at their core, a machine learning system. Given a query, in the form of a text string, its goal is to return links to the web pages that will most likely match the intent of the user. The results have to be accurate, fast, and up to date, and the system needs to be always available. It is constantly updated on new data collected from the web, and it is adapted to the preferences of the user. Some movie streaming websites contain a recommendation system that aggregates what other users watched, combined with the history of a specific user, to try to propose what the user could want to watch next. Its goal is to keep the user engaged (some could think it performs too well, as many of us can attest after too many hours of watching a series). It needs to be constantly available, serving huge amounts of data, and constantly adapting to the changing preferences of the users. Different challenges are encountered by digital financial and payment services. They need to constantly monitor transactions to flag inappropriate and fraudulent ones, with very little delay and high accuracy; rejecting good transactions annoys users, while accepting bad ones is costly. These systems need to have several failsafe mechanisms and be constantly monitored for performance and delays.

Version Control

Version control is the management of changes to documents, computer programs, and other collections of information. The reason we need versioning is to track changes, be able to revert to previous working solutions, and be able to coordinate updates. We are already familiar with **Git** for version control, and many tools aimed at the machine learning process either use it or are based on it. We usually identify the changes via numbers or letters, called revisions or revision numbers. An initial version of a set of files or code could be "revision 1." After a change is made, we will have "revision 2," and so on. Using versions, and a **version control system**, allows us to keep track of changes (specifically what was changed when and by whom), compare different versions, and synchronize changes between different parts of an application.

Git
GitHub is a popular platform based on Git, hosting many open-source projects.

Version control system
This keeps track of changes, including what was changed when and by whom, and can compare different versions.

Versioning Code

When considering the code purely as a "programming language," there are two different kinds of code in the case of an ML system: implementation code and modeling code. The implementation code could be glue code, the code used to access APIs, or system integration code which connects the ML system to the other applications. Modeling code is used for model development. In some cases, the code can be written in several different languages. This code needs to be versioned for releases, and the releases tested. Additionally, code has dependencies that need to be updated, and it will interact with other pieces of code. So far, this is exactly what happens in any normal software development process, where the concept of infrastructure as code (IaC) has been developed to solve the problem of environment drift in the release pipeline. However, as we have noticed, the ML system is not only the code to generate the model and the code to wrap it and connect it to other services: It includes the data used to generate the models, and the models themselves, which all need versioning.

Versioning Data

The data used to train the model can change. The **metadata** (format, names, order, and number of columns) can change, values can be dropped or added, precision for some values can change, or the statistical characteristics compared to the new data can change too much compared to the data used to originally create the model. To be able to generate the same model over and over again, a requirement for reliable deployment, we need to make sure we are using the same data that we used when we shipped. This means versioning the data. As noted, the data includes both the data and the metadata. The main issue with the data is size: The data used to train a model can span to a few megabytes, or even several petabytes in some cases. Storing a few megabytes in a normal version control system isn't a problem; however, storing gigabytes (or more) is. This has given rise to several systems to store larger files, from the open source add-ons to Git (like Git Large File Storage

Metadata
This tells us about the nature of a certain column (or feature) in our dataset.

(LFS) (GitHub, n.d.-c) and Data Version Control (DVC) (DVC, n.d.)), to several closed source solutions. The two main issues of versioning the data are storage requirements for potentially huge amount of data, and the task of keeping the metadata matched with the requirements of the model.

Versioning Models

The models, and other artifacts generated during the model development, also need versioning, so that we can match them with the algorithms used to generate them, the code that supports them and that uses them, and the data used to train them. The storage requirements of models are not as problematic as the ones for the data, even if models can still require hundreds of megabytes, posing a challenge for normal version control systems. Git has a maximum file size limit of 100mb, but we have already mentioned Git LFS as a workaround for that (GitHub, n.d.-c).

Reproducible Model Training

It should be possible to rebuild the model on demand starting from a specific version of the data and the code used to generate the model, using the specific version we desire. However, since this takes time and resources, we want to store the version of the model together with the rest. Versioning ensures that we can always reproduce the same results at different points in time and on different machines, starting from the same version of data and code.

Deployment

Once developed and trained, the model needs to be deployed for it to be of any use to the organization. We have two main possibilities.

- Embedded model**:** The model as an artifact (the actual file) is built and packaged together with the application using it. From now on, it is considered a part of the application resources, the same way a background image for a website would be.
- Model deployed as a service**:** The model is wrapped in a service that can be deployed independently of the consuming applications. This allows for decoupling, but can increase latency, and the applications need to run some sort of remote invocation.

In both cases, the models need to be stored and once built, delivered to the appropriate infrastructure. Deploying as a service adds some complexity, but we can think of the service as an application: In such a case, the model artifact is stored and deployed embedded with the serving application. The main considerations for deployments are practically identical to the ones in traditional CI/CD pipelines for software development, including integration tests and system tests. Still, model deployment can present different complex scenarios.

Multiple models

We can sometimes have more than one model for the same task, or several models working together for the same task. We can choose to deploy the models as separate services, or accessing multiple models with a single API call. The first possibility offers more flexibility, but the consuming applications then need to be extended, while the second scenario allows us to change the number and type of models without rewriting the rest of the application.

Shadow models

In some critical scenarios, it can be desirable to have several versions of the same model in production side by side. This allows us to make sure that the new models, that we expect to perform better than the old ones, actually do. To ascertain it, we gather data on the shadow model behavior before moving it to active production, a procedure similar to testing, but on live data: This way, we can make sure the behavior of the new models is as expected before going live.

Competing models

In a more complex scenario, we can keep multiple versions of the same model in production to find out which one is best, like an **A/B test**. This adds complexity in the form of infrastructure and routing to make sure that the traffic reaches the right models, and to make sure we have enough data to make decisions on what model to keep.

A/B test
This test consists of a randomized experiment with two variants, A and B.

Online learning models

The models we discussed so far are developed, trained, and deployed. Online models get updated constantly and can continuously improve their performance with new data, learning while in production. This presents extra complexity: the models aren't static artifacts, they won't yield the same results. We need to make sure the model performance doesn't degrade, that the data used for training is acceptable, and we need versioning of updated models and production data to be able to revert to a version with good performance in case something goes wrong. It is a quite advanced scenario, rarely used outside of very specialized applications, usually in very big organizations. To support more complex deployment scenarios, it is advisable to use elastic infrastructure that can scale on demand. This involves using cloud providers which will be covered in a later section.

Testing and Quality

In software engineering, testing allows us to increase our confidence that our system behaves correctly, and allows us to be warned when something breaks. Testing in an ML environment has a bigger scope than in traditional software engineering, since it covers data and training, in addition to the integration with other components.

- Data: We can add tests to validate input data against the expected schema, or to validate our assumptions about valid values. We want to be sure the data being fed to the model corresponds to the kind of data the model was trained for. This is even more important if we expect the model to be constantly updated on new live data.
- Component integration: We want to make sure the different components of a system work well together, behave as expected, and communicate with each other using the expected protocols and contracts. We also want to make sure that the behavior of the components in production is the same as the ones in development, i.e., validating with the same datasets should get us the same results. This is not as obvious as it seems, and any mismatch should be investigated.
- Model quality: The performance of an ML model is non-deterministic, but we can collect and monitor the relevant metrics to evaluate the model's performance. We will delve into this more, but in general we want to make sure that the relevant metrics stay within an acceptable range, above all in case of retraining.
- Bias and fairness: We want to make sure that the model behaves in a way that is aligned with the goal of the organization. This is out of scope from the present topic, though.

Monitoring and Observability

Once the model is live, we need to understand how it performs in production, and close the feedback loop to the development process. This is the moment we can gain reliable data on what we have developed so far. The main tools are the standard tools for production software monitoring, with a specific application to the machine learning process. Production software uses tools for log aggregation and metric collection, to capture the data from a live production system: KPI, software reliability, performance, debugging information in the case of faults, and other indicators that something unexpected is happening. In general, we want to be notified when something goes wrong or when something strange happens to allow us to investigate. In a machine learning system in production, we want to specifically keep track of

- model inputs. These are the data that is being sent to the models, to allow us to keep track of the model-serving skew.
- model outputs. Seeing what predictions, recommendations, results are the models giving to these inputs, to understand how the model is behaving on real data. This can include the decisions taken based on the outputs, since in some cases we want to define exceptions where our system will overrule the outputs of the model in some specific cases.
- model interpretability outputs. These metrics allow further investigation to understand how the models are making predictions to identify potential overfit or bias not found during training. As an example, Amazon introduced a machine learning tool to screen the resumes of applicants, that seemed to work quite well, until they realized it was strongly biased against women, since in the training dataset, very few women were hired. Amazon has since stopped using the tool (Dastin, 2018).
- user action and rewards. We want to keep track of what the users do once they receive the outputs. Do they buy what was presented to them, do they watch the movie, do they pay on time when given credit? This is very important to understand if we are actually solving the right problem.

- model fairness. As noted above, we want to keep track of the behavior of models regarding race, gender, age, etc. We want to know if the model is behaving in a way that is not aligned to our values.
- model computational performance. As for all software systems, the reaction time can be critical. For some user interaction, a result delay of a few seconds is the same as no result at all. We want to keep track of how long models take to react, the CPU, and memory load in case we need to scale.

Collecting, monitoring, and observing data is even more important in the case of multiple models deployed in production. To assess a shadow model, we can perform A/B tests or run multiple experiments. The collected data is essential to close the data feedback loop: Using more real-world data or adding **humans in the loop (HITL)** to analyze the new data resulting from production usage, we can create new datasets to generate new, and hopefully improved, models. It allows us to learn to adapt our models based on their behavior on real production data, allowing for continuous improvement.

Humans in the loop (HITL)
This is a model that requires human interaction, as opposed to a closed-loop model that is without human interaction.

Model Drift

Data can change over time, making the performance of our models degrade if they do not change with the data. We call this problem **model drift** (Brownlee, 2019).

Model drift
This is the degradation of the performance of the model due to either change in the nature of the current input data or changes in the training data.

We can identify two basic kinds of model drift:

- Concept drift: happens when the statistical properties of the desired output itself change. As an example, the predicted spending of customers won't be accurate if there are promotions not included in the model (Wikipedia, 2020d).
- Data drift: occurs when the statistical properties of the input change, compared to the data used for training. A simple example with seasons: a model of beverage sales trained in the summer will not work without changes in the winter.

The change in data can take different forms, and we can conceptually model it as

- gradual change over time. Some trend affects the data, for example people moving to cities from the countryside.
- recurring or cyclical change. Seasons and days of the week are a typical example, as are holiday season, end of quarter, etc.
- sudden or abrupt change. Think of the impact of the Covid-19 pandemic on flights and the economy in general, or of the introduction of the iPhone on the sales of other kind of smartphones.

To address model drift, we can consider severalaspects (Zliobaite, 2010).

- Future assumption: Assumptions about the future data sources, in what way they can differ from what we have, and how to both acquire data and integrate it in the system
- Change type: Identifying possible change patterns, what changes we can predict, and how we plan to adapt to them. **Seasonality** is the easiest to think about, underlying trends due to unknown causes the hardest.

- Learner adaptivity: Choosing how to adapt the models based on the change type and the future assumption
- Model selection: A criterion to choose a particular parametrization of the selected learner at different time steps

Seasonality
This is the presence of variations that occur at specific regular intervals less than a year, such as weekly, monthly, or quarterly.

Possibilities of action and reaction to model drift include

- doing nothing: We assume that the data does not change and keep a static model. At times, doing nothing performs better than doing something.
- periodically re-training: We can periodically update your static model with more recent historical data. We can update the model every month with new data, then use the new model as a static model. In some cases, we can use only a small sample of new data, or a sliding window.
- weighing data: Some algorithms allow us to weigh the importance of input data, so that we can give more weight to more recent data.
- detecting and choosing a model: In some cases, it is possible to design systems that detect changes and choose a specific, different model to make predictions. This can be appropriate in systems designed to properly react to sudden changes without human intervention. Consider the case of a trading algorithm that needs to react to a crisis without being rewritten.
- data preparation: In cases of time series problems, the data are already expected to change over time, and can be partially prepared to remove the systematic changes over time, if those changes can be modelled. Technically, systematic change is not really a model drift, since we can plan for it in advance, but some changes are not systematic, a challenge in using models in the financial and other industries.

When Things Go Wrong

Now that we have outlined the steps to be considered when bringing a model to production and the potential challenges, let's review the reason behind all of these steps: We are trying to minimize problems, and make it easy to recover when things go wrong. In production, something will always go wrong, sooner or later, and we need to both minimize the effects and be able to recover. What can happen, and how can the elements outlined above help? Well, if we stay in production long enough, we will have to retrain the model because of model drift, new challenges, changing business requirements, new data, or other unforeseen events. We need versioning for this, and we need to make sure that the version includes the data used to train the models. Due to outages, or malicious attacks, part of the infrastructure can get lost, but we still need to be able to regenerate the models and we need to make sure the retrained model behaves as we expect. To know when to implement the measures we just learned about, we need monitoring, and also to know when we need more resources because the performance is too low. After updating a model, we need to be able to know if the new model behaves as expected (monitoring) and to be able to quickly roll back to a previous version in case it doesn't (versioning). In the case of online learning (a constantly updated and retrained model), we need to make sure that the performance does not degrade, and we need to be able to roll back to a known working version in case it happens: again, monitoring, versioning, reproducibility.

5.3 MLOps and DataOps

As we have seen in the previous section, putting and keeping a machine learning system, including the model, in production presents several challenges and is a natural counterpart in the domain of data science to the DevOps of traditional software development. DevOps is commonly used in the development and operation of large-scale software systems, providing benefits such as shorter development cycles, increased speed of deployment, and dependable releases. The two main concepts used to achieve this are continuous integration (CI) and continuous delivery (CD). A machine learning system is a software system, meaning we can adapt these techniques using the following criteria.

Team Skills

Data scientists are part of the team and are not always experienced engineers, so their code often needs some work to reach production quality.

Development

Machine learning is experimental and non-deterministic. It is necessary to experiment with different features, algorithms, configurations, datasets, and problem definitions to find out what works as quickly as possible. It is challenging to keep track of what did or did not work and simultaneously maintaining reproducibility and code reusability.

Testing

As noted above, testing an ML system includes elements not present in other software systems. In addition to typical unit and integration tests, you need data validation, trained model quality evaluation, and model validation.

Deployment

Machine learning systems can require deployment in a multi-step pipeline to automatically retrain and deploy models, adding complexity and the necessity to automate steps that before deployment are manually performed by data scientists.

Production

The performance of machine learning systems can be affected by the quality of coding and changes in data profiles. Models can decay in more ways than other software systems, and it is necessary to keep track of this with statistics and monitoring, sending notifications or automatically rolling back to known working versions.

As such, machine learning and other software systems are similar in the requirements for continuous integration of source control, unit testing, integration testing, and continuous delivery of the software module or the package. However, they differ in a few areas. CI is no longer only about testing and validating code and components, but also testing and validating data, data schemas, and models. At the same time, CD is no longer about a single software package or a service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service). We can also add the property of continuous training (CT). CT is a new property, unique to ML systems, that's concerned with automatically retraining and serving the models (Google, 2020a). These different and added requirements are leading the creation of two additional practices for data intensive operations: DataOps and MLOps. DataOps (a compound of "data (analytics)" and "operations") is the practice of using automated, process-oriented methodologies in data teams to improve the quality and reduce the cycle time of data analytics, covering the entire data lifecycle from data preparation to reporting. Toph Whitmore, principal analyst at Blue Hill Research, defined it as follows (Vorhies, 2017):

1. Measure progress and performance at every step of the data flow.
2. Define rules about what the data and metadata are.
3. Have humans in the feedback loop to make sure assumptions hold.
4. Automate as many stages as possible.
5. Identify bottlenecks.
6. Include governance compliance in the process, including data control, data ownership, transparency, and tracking.
7. Design for growth and extensibility, accommodating for increasing volume and variety of data.

MLOps (a compound of "machine learning" and "operations") is a practice for collaboration and communication between data scientists and operations professionals to help manage the production machine learning lifecycle described in the previous section, covering

- deployment and automation,
- reproducibility of models and predictions,
- diagnostics,
- governance and regulatory compliance,
- scalability,
- collaboration,
- business uses, and
- monitoring and management.

Other names and proposed concepts are AIOps, ModelOps, and DLOps, which are respectively responsible for using AI to automate IT operations, automating ML models, and MLOps specifically for deep learning. They can all be considered subcategories of MLOps. Given the recent explosion of interest in and usage of machine learning systems, several tools have been developed to support MLOps, including Airflow, Luigi, Argo, Kubeflow, MLFlow, Michelangelo, and many others. They partly overlap in features, partly solve different problems in different ways, and partly impose a way of solving problems that can either match well with the workflow of our organization or present some mismatch. We will cover in depth MLflow for model tracking, Kubeflow for machine learning pipelines, and Michelangelo for the complete machine learning end-to-end process. These solutions integrate into the existing infrastructure of the organization, in some cases with only small changes and in others, with adaptations to the implicit model of the tools.

Model Tracking With MLFlow

MLFlow allows the automation of model development and tracking so that the optimal model can be selected (MLflow, n.d.). We can log parameters, attributes, and performance metrics, using them to identify the models that fit particular criteria. Its main feature is that it enables you to keep track of your ML experiments, amongst others, by logging parameters, results, models, and data for each trial. MLflow is designed to be agnostic regarding machine learning libraries, algorithms, deployment tools, or languages. It is aimed to be easily added to an existing machine learning code, sharing code inside the organization using any machine learning library, allowing others to run it. While mostly aimed at the model selection part of the lifecycle, it can support the complete machine learning lifecycle when used with other tools like **Airflow**. The design philosophy of MLflow has a modular and API-based design, with functionality divided into four parts: tracking, projects, models, and registry. MLflow tracking is a centralized place for obtaining the details of the model, a sort of meta-store. The client applications communicate with the tracking server via the HTTP protocol, with APIs in Python, REST, R, and Java. The tracking server captures details for the model and uses backend stores to log

Airflow
Apache Airflow is an open-source workflow management platform.

- logging parameters,
- code versions,
- metrics,
- artifacts (model and data files),
- start and end time of the run, and
- tags and notes.

MLflow project

MLflow project is organized and packaged code to support the reproducibility of a model. It uses a YAML file, named MLProject, that describes the requirements of the machine learning project.

A simple example of an MLProject

Code
```
name: sklearn-demo
conda_env: conda.yaml
entry_points:
  model_run:
    parameters:
      max_depth: int
      max_leaf_nodes: {type: int, default: 32}
      model_name: {type: string, default: "tree-classification"}
      run_origin: {type: string, default: "default" }
    command: "python model_run.py -r {max_depth} {max_leaf_nodes} {model_name}"
```

In this example, we are defining a decision tree by

1. setting the name to `sklearn-demo`.
2. declaring we'll use the **Anaconda** environment `conda.yaml`.
3. defining some parameters: `max_depth`, required, the maximum depth of the tree, as an integer; `max_leaf_nodes`, an integer with 32 as default value, the maximum number of leaf nodes of the tree; `model_name`, a string with the name of the model; `run_origin`, where we want to start.
4. the command that will be run, the Python interpreter on the `model_run.py` file.

Anaconda
It is a distribution of the Python and R programming languages for scientific computing.

MLflow models

MLflow models define a format for packaging machine learning models that can be used in a variety of downstream tools, like a REST API or Apache Spark. The format allows saving the model using different "flavors" that correspond to different tools. "Flavors" are a key concept for MLFlow, a convention to allow deployment tools to understand the models, making it possible to work with any ML library without having to integrate the tools in the library itself. They allow us to

- utilize the same memory format for different systems,
- avoid the overhead of cross-system communication (serialization and deserialization), and
- provide common shareable functionalities.

MLFlow has built-in flavors for many popular machine learning algorithms and libraries, such as H2O, Keras, MLeap, PyTorch, Scikit-Learn, MLlib, Tensorflow, ONNX (Open Neural Network Exchange), MXNET gluon, XGBoost, and LightGBM. Additionally, it offers the possibility to create custom flavors using Python (MLflow, n.d.).

Model registry

The model registry is aimed at solving the problem of machine learning model management, allowing management of the full lifecycle of the machine learning model providing, using the following concepts.

- Model: A model is created from an experiment or run that logged via specific MLflow logging methods. After being logged, the model can then be registered with the model registry.
- Registered model: It is then registered in the model registry with a unique name, container versions, associated transitional stages, model lineage, and other metadata.
- Model version: Registered models can have one or many versions. When added, a model starts at version 1, with every new model registered to the same model incrementing the version number.
- Model stage: Every distinct model version can be assigned to one stage, like staging, production or archived, or something custom. Models can be transitioned to different stages.
- Annotations and descriptions: It is possible to annotate the whole model, and each version, including the description and any relevant information.

MLFlow can be simply be installed as such:

Code
```
pip install mlflow
```

A simple pipeline would look like this:

Code
```
from mlflow import log_metric, log_param, log_artifact

if __name__ == "__main__":
   # Log a parameter (key-value pair)
   log_param("param1", 5)

   # Log a metric
   log_metric("foo", 1)
   log_metric("foo", 2)
   log_metric("foo", 3)

   # Log an artifact (output file)
   with open("output.txt", "w") as f:
       f.write("Hello world!")
   log_artifact("output.txt")
```

The code above

- logs a parameter `param1` with value 5;
- logs metrics called **foo** with values 1, 2, and 3;
- opens an existing file (an artifact), `output.txt`, to write to it; and
- logs the artifact for the run with `log_artifact`. This is what we mostly care about, keeping track of the changes for every time the job runs.

Foo
Foobar, foo, bar, and similar are used as placeholder names in computer programming.

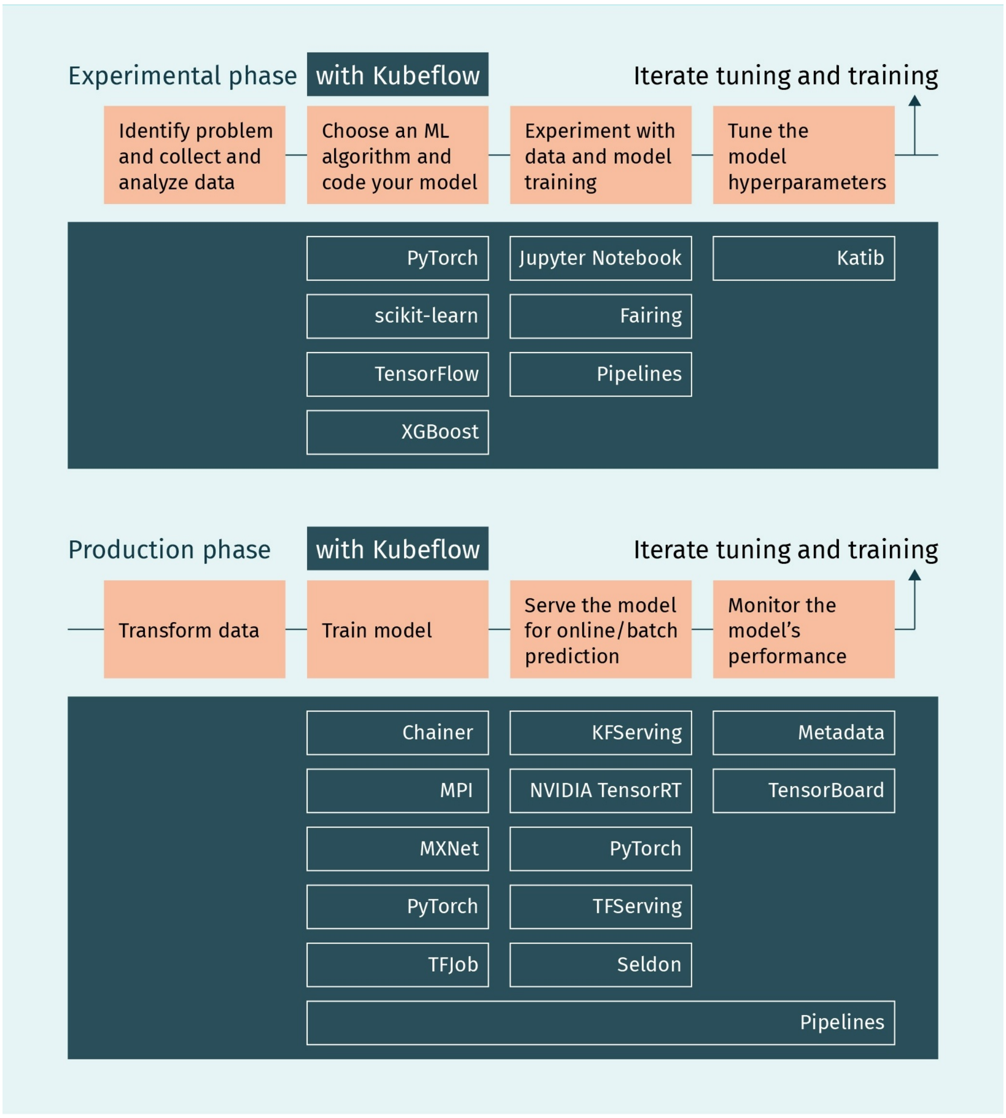
This is clearly a useless example, hence the **"Hello, World!"**, to just give an idea of how to programmatically annotate the steps of the process in a Python script (GitHub, n.d.-b).

"Hello, World!"
This is a very simple program often used to illustrate the basic syntax of a programming language.

Pipeline Management With Kubeflow

Kubeflow was started by Google as an open-source platform for running TensorFlow. It is a machine learning platform that manages deployments of ML workflows on Kubernetes and is a scalable and portable solution (Kubeflow. n.d.-a). It is aimed at data scientists building and experimenting with data pipelines, and for deploying machine learning systems to different environments in order to carry out testing, development, and production-level service. It is a multi-cloud, multi-architecture framework that runs entire ML pipelines, and offers specific tools for all the steps we covered in the model creation pipeline, as well as the deployment pipeline.

Figure 50: Model Development and Production With Kubeflow


Experimental phase
with Kubeflow
Iterate tuning and training
Identify problem and collect and analyze data
Choose an ML algorithm and code your model
Experiment with data and model training
Tune the model hyperparameters
PyTorch
scikit-learn
TensorFlow
XGBoost
Jupyter Notebook
Fairing
Pipelines
Katib
Production phase
with Kubeflow
Iterate tuning and training
Transform data
Train model
Serve the model for online/batch prediction
Monitor the model's performance
Chainer
MPI
MXNet
PyTorch
TFJob
KFServing
NVIDIA TensorRT
PyTorch
TFServing
Seldon
Metadata
TensorBoard
Pipelines

Source: Kubeflow, (n.d.-b).

As you can see from the diagram, Kubeflow integrates in commonly used tools (PyTorch, scikit-learn, Jupyter Notebooks, TensorBoard, etc.) and adds some components specific to KubeFlow. One component is the services for spawning and managing Jupyter Notebooks, interactive data science, and experimenting with ML workflows. It permits the sharing of notebooks across the organization and the creation of notebooks in pods or containers in the cluster, instead of locally. Kubeflow integrates admin controls to allow for standard notebook images and to set up role-based access control (RBAC), as well as secrets and credentials to manage which teams and individuals can access the notebooks. Kubeflow Pipelines offers a platform for building, deploying, and managing multi-step ML workflows based on Docker containers. In Kubeflow, a pipeline is a description of an ML workflow, including all components and how they combine. It includes the definition of inputs and outputs of each component. Every component of a pipeline is a Docker image with self-containing code that performs one step of the pipeline. A component can be responsible for data preprocessing, data transformation, model training, and so on (MLflow, n.d.).

The pipelines are described with Python code. What follows is a toy pipeline for a function adding two numbers. Please note that the Python SDK for Kubeflow is `kfp`. We define a simple function `add`, we add it to a component, we then create a pipeline called `calc_pipeline` (using the `@dsl.pipeline` decorator), set the arguments we will pass to the pipeline (hence to the function), and finally create a run for the pipeline with the following code:

Code

```
kfp.Client().create_run_from_pipeline_func().
```

Code

```
import kfp
import kfp.components as comp
import kfp.dsl as dsl
# we define a very simple function, adding 2 numbers
def add(a: float, b: float) -> float:
  '''Calculates sum of two arguments'''
  return a + b
add_op = comp.func_to_container_op(add)

@dsl.pipeline(
  name='Calculation pipeline',
  description='A toy pipeline that performs arithmetic calculations.'
)
def calc_pipeline(
  a='a',
  b='7',
):
   result_task = add_op(a, 4) #Returns a dsl.ContainerOp class instance.

#Specify pipeline argument values
arguments = { 'a': '7', 'b': '8'}
#Submit a pipeline run
kfp.Client().create_run_from_pipeline_func(calc_pipeline, arguments=arguments)
```

Kubeflow offers several components that you can use to build your ML training, hyperparameter tuning, and serving workloads across multiple platforms. As we can see, it offers solutions to many of the issues we discussed in the previous section.

End-to-End Machine Learning Lifecycle With Michelangelo

Michelangelo (Hermann & Del Balso, 2017) was built in 2015 by Uber to tackle what they named "hidden **technical debt** in machine learning systems" (Sculley et al., 2015). It was built to solve the problem of custom one-off systems tightly coupled with machine learning models, an approach that did not scale well, as we have previously seen. Michelangelo relies on transactional and logged data and supports offline (batch) and online (streaming) predictions. We use offline predictions to run the model on a whole dataset at once, getting all results in batch when done, and we use online predictions to respond to queries as they come. Offline predictions are based on containerized **Spark** jobs for batch predictions, while online predictions are served in a prediction cluster, allowing for a load balancer to distribute the load to several machines. Load balancing is the process of distributing a set of tasks over a set of resources, with the aim of making their overall processing more efficient. Every experiment stores metadata relevant to model management (e.g., run-time statistics of the trainer, model configuration, lineage, distribution and relative importance of features, model evaluation metrics, standard evaluation charts, learned parameter values, and summary statistics). With Michelangelo it is possible to deploy multiple models in the same serving container, allowing to transition from old to new model versions and side-by-side A/B testing of models. Michelangelo supports both online and offline models. The data preparation pipelines push data into the feature store tables and training data repositories. They use Apache Kafka (an open-source stream processing software) and connect to a **Hadoop** File System data lake. The data preparation uses either Apache Spark or SQL. The model training happens in batch, using one of the many supported models: decision trees, linear and logistic models, unsupervised models (k-means), time series models, and deep neural networks. A model configuration specifies the model type, hyperparameters, and data source reference, and computes resource requirements. It is then run on a cluster. After the model is trained, performance metrics are saved into a model evaluation report and the original configuration, the learned parameters, and the evaluation report are saved back to our model repository for analysis and deployment. Hyperparameter search is supported for all model types. For every trained model, a versioned object is stored in the model repository in Cassandra (a distributed, wide column store, NoSQL database management system) for evaluation. Reports can be visualized in a dashboard, with standard accuracy metrics and feature reports. Models can then be deployed in several ways: offline; in a container then run in a Spark job to generate batch predictions; as an online prediction service cluster; or embedded as a library embedded in another service. Once models are deployed and loaded by the serving container, they are used to make predictions based on feature data loaded from a data pipeline or directly from a client service. Multiple models can be deployed to a given container, and monitoring of metrics and performance is logged. The platform uses Spark's machine learning pipeline serialization with an additional interface for online serving that adds a single-example (online) scoring method, lightweight and capable of handling **service-level agreements (SLA)**, necessary in critical cases such as for fraud detection and prevention.

Technical debt
It reflects the implied cost of additional rework caused by choosing an easy and limited solution as opposed to a more sustainable approach that would take longer to implement.

Spark
Apache Spark is an open-source distributed general-purpose cluster-computing framework.

Hadoop
Apache Hadoop is a collection of open-source software utilities that facilitates using a network of many computers to solve problems, involving massive amounts of data and computation.

Service-level agreement (SLA)
This is a commitment between a service provider and a client.

Concluding Remarks

We have covered the challenges of bringing machine learning to production from the point of view of MLOps. The tools we have explored have different strengths and different approaches. MLFlow, for example, is tightly integrated with Spark (it is developed by the same company). It is mainly a model management library, with limited options for deployment, which needs to happen using Airflow. Kubeflow is a more integrated solution, build on Kubernetes and integrating several open-source building blocks. It requires separate configuration of many of the elements needed. Lastly, Michelangelo is an example of an end-to-end solution, integrating most of the components with structured access to the ones that are not integrated. As such, it imposes some choices on us, removing some flexibility and adding complexity to the system. Each of these systems requires the organization to set up the necessary resources, both hardware and software, and requires an experienced support team.

5.4 Cloud-Based Solutions

Cloud infrastructure
Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user.

We have already discussed some of the challenges posed by bringing machine learning to production, the emergence of MLOps, and tools that solve some, most, or all of these challenges. These tools assume the use of an already existing infrastructure in the organization, and in some cases permit the use of hybrid solutions, accessing **cloud infrastructure** for some tasks. Building and maintaining an own infrastructure presents some problems and costs. Among others, it requires in-house knowledge, utilizes internal resources, and requires maintenance of the infrastructure on the hardware and software levels, as well as scalability. Some organizations do not want to incur those costs, and the emergence of machine leaning cloud services addresses this need. Using machine learning cloud services, it is possible to build, monitor, and deploy working models, and use them for predictions and insight, all with a relatively small team.

Machine Learning as a Service (MLaaS)

Remote procedure call (RPC)
This is when a computer program causes a procedure to execute in a different computer in a network.

Machine learning as a service (MLaaS) is an umbrella definition of various cloud-based platforms that cover most infrastructure issues we considered for MLOps: data pre-processing, model training, model evaluation, deployment, and monitoring. The output can then be bridged to the internal IT infrastructure through REST APIs and other forms of internal communication like **RPC** and other protocols. In 2020, the leading cloud MLaaS providers are Amazon Machine Learning services, Azure Machine Learning, Google Cloud AI, and IBM Watson (IBM, n.d.-c), all aiming to allow for faster model training and deployment (AltexSoft, 2019).

Amazon Machine Learning services are available on two levels: predictive analytics with Amazon ML, and the SageMaker tool mostly aimed at data scientists (Amazon, n.d.-b). Amazon Machine Learning for predictive analytics is a mostly automated solution, aimed at deadline-sensitive operations. It can load data from multiple sources, for example, Amazon RDS, Amazon Redshift, and CSV files. Data preprocessing operations are performed automatically. Amazon ML tries to identify which fields are categorical and which are numerical, and it doesn't ask a user to choose the methods of further data preprocessing such as dimensionality reduction. However, the prediction capacities of Amazon ML are limited to binary classification, multiclass classification, and regression. It does not support unsupervised learning methods, and a user must select a target variable to label it in a training set. Since it is aimed at non-data scientists, a user isn't required to know any machine learning methods; instead, they are chosen automatically from the provided data. It provides a fully automated, yet limited, solution. For more advanced usage, more flexibility and complexity, there's SageMaker. Amazon SageMaker is a machine learning environment aimed at providing tools for quick model building and deployment, integrating Jupyter for data exploration and analysis without server management hassle (Amazon, n.d.-b).

Machine Learning With Amazon SageMaker

To explore how cloud solutions can provide all the tools to build an end-to-end pipeline for machine learning, we will now concentrate on Amazon SageMaker. The offerings from other providers tend to be similar in their capabilities, with differences in the specifics. Amazon SageMaker is a fully managed service that provides all the tools to build, train, and deploy machine learning models. SageMaker allows us to execute all the steps of the machine learning pipeline (Amazon, n.d.-c). Let's quickly review them: understanding the problem, collecting and preparing the data, performing exploratory data analysis, experimenting with features and models, training and selecting the final model, keep versions, deploy the model, monitor performance, and integrate with other systems.

Prerequisites

Based on Amazon AWS, using SageMaker requires having or setting up an AWS account, then creating an IAM administrator user and group (Amazon, n.d.-d). AWS is very powerful and not always intuitive, and it can take some time to find out how to do things properly. Bigger organizations will have dedicated processes and teams for this. But if we are setting it up ourselves, we need to make sure to have an IAM user with administrative rights. Managing AWS services is a topic that could fill whole books, so it is highly recommended to get at least a bit familiar with it (Amazon, n.d.-e).

Problem understanding

At the moment, no automated system can really help us with problem understanding. That would be an application for a quite advanced AI. SageMaker offers some automatic tools, like SageMaker Autopilot, but they are limited to few domains: autopilot, regression, binary classification, and multiclass classification.

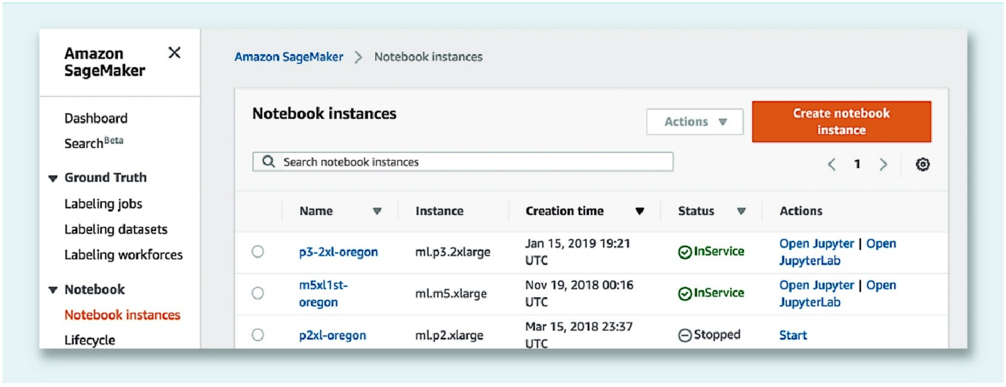
Data collection, cleaning and wrangling

A step we have not considered so far, which is essential when dealing with data in an organization, is where the data come from and where they are stored. As a fully managed solution based on the many services present in AWS, SageMaker offers or integrates functionalities for data storage: this step involves setting up a **data lake**. Generally, we can access data from several sources, but in the AWS environment, most of the time the raw data will be initially stored to an S3 bucket. Amazon Simple Storage Service (S3) provides object storage through a web service interface. We can also save preprocessing code on S3 and point SageMaker to it. In SageMaker, the data is preprocessed in a **Jupyter** notebook, in a notebook instance. We use notebooks to fetch the dataset, explore it, and prepare it for model training. To start, we need to launch a notebook instance in SageMaker.

Data lake
A data lake is a system or repository of data stored in its natural (raw) format, usually object blobs or files.

Jupyter
This is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

Figure 51: SageMaker Administrative Console


Amazon SageMaker

Dashboard
Search^{Beta}

Ground Truth
Labeling jobs
Labeling datasets
Labeling workforces

Notebook
Notebook instances
Lifecycle

Amazon SageMaker > Notebook instances

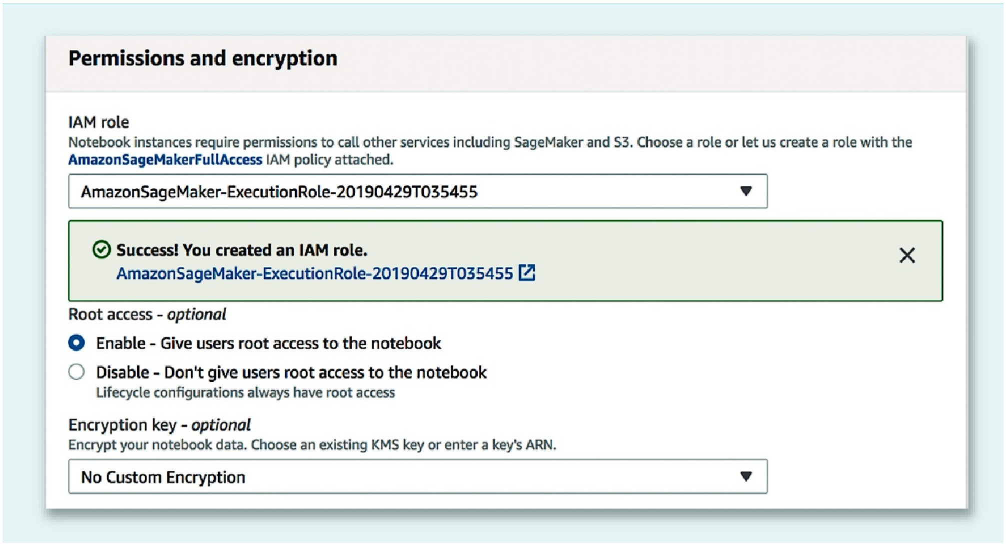
Notebook instances

	Name	Instance	Creation time	Status	Actions
○	p3-2xl-oregon	ml.p3.2xlarge	Jan 15, 2019 19:21 UTC	InService	Open Jupyter | Open JupyterLab
○	m5xl1st-oregon	ml.m5.xlarge	Nov 19, 2018 00:16 UTC	InService	Open Jupyter | Open JupyterLab
○	p2xl-oregon	ml.p2.xlarge	Mar 15, 2018 23:37 UTC	Stopped	Start

Source: Pedori (2020).

We can then select name, type of instance, and other settings. In AWS, it is always necessary to create an IAM role. In our specific case, it is necessary to grant some level of access to the notebook.

Figure 52: Granting Access to IAM Roles


Permissions and encryption

IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the **AmazonSageMakerFullAccess** IAM policy attached.

AmazonSageMaker-ExecutionRole-20190429T035455

Success! You created an IAM role.
AmazonSageMaker-ExecutionRole-20190429T035455

Root access - *optional*
[✓ Enable - Give users root access to the notebook]
Disable - Don't give users root access to the notebook
Lifecycle configurations always have root access

Encryption key - *optional*
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption

Source: Pedori (2020).

Once set up, we can access the notebook instance and open Jupyter. From the notebook, we can perform all the data preprocessing steps, using AWS Glue for executing a SparkML feature pre-processing and post-processing job. AWS Glue is a serverless data preparation service enabling us to extract, clean, enrich, normalize, and load data, and allowing for ETL operations in a graphical interface. SparkML is a high-level API that helps users create and tune practical machine learning pipelines from the Apache Spark project. We can set up several steps for data preprocessing, programmatically or manually. The standard Python client for AWS services is called boto (boto, 2020). The following code, job creation omitted, shows how to run a job using AWS from a notebook:

Code

```
import time
import boto3

glue = boto3.client('glue')

# Run the job in AWS Glue
job_name='preprocessing-cars'
response = glue.start_job_run(JobName=job_name)
job_run_id = response['JobRunId']
print('{}\n'.format(response))

# Check on the job status every 30 seconds
job_run_status = glue.get_job_run(JobName=job_name,RunId=job_run_id)
['JobRun']['JobRunState']
while job_run_status not in ('FAILED', 'SUCCEEDED', 'STOPPED'):
   job_run_status = glue.get_job_run(JobName=job_name,RunId=job_run_id)
   ['JobRun']['JobRunState']
   print (job_run_status)
   time.sleep(30)
```

Exploratory data analysis (EDA)

Using a normal Jupyter notebook, the EDA steps are the same as would be performed on a local machine or on-premise resources, using the same Python code. This step does not need any adaptation; the only difference is that the notebook you work with is hosted on AWS by SageMaker.

Experimentation, training, and versioning

Where the tools provided by SageMaker begin to shine is when we start experimenting with models, features, and hyperparameters. We run everything as a training job, either from the UI or from the Python SDK. SageMaker includes several built-in models that we can apply to a problem (Amazon, n.d.-f). They cover supervised learning, unsupervised learning, textual analysis, and image processing. The complete list of currently available algorithms in 2020 is BlazingText algorithm, DeepAR forecasting algorithm, factorization machine, image classification algorithm, IP insights, k-means algorithm, k-nearest neighbors (k-NN) algorithm, latent dirichlet allocation (LDA) algorithm, linear learner, neural topic model (NTM) algorithm, Object2Vec algorithm, object detection algorithm, principal component analysis, random cut forest (RCF), semantic segmentation algorithm, SageMaker sequence to sequence (seq2seq), and XGBoost (eXtreme gradient boosting). It is also possible to use your own algorithm, packaging it in a Docker container, by either extending a prebuilt Docker image or adapting/creating your own. For feature selection and hyperparameter tuning, SageMaker is well integrated with SparkML, using it for most tasks. One of the most important contributions of SageMaker is “Experiments,” letting you organize, track, compare, and evaluate your machine learning experiments. SageMaker experiments introduce the concept of trial and experiment. A trial is a collection of training steps involved in a single training job, including preprocessing, training, and evaluation, enriched with metadata. An experiment is a collection of trials, a group of related training jobs. In SageMaker experiments we create experiments, populate them with trials, and run analytics across trials and experiments. It is available both from the UI and through a Python SDK containing logging and analytics APIs. Additionally, experiments are integrated in Sagemaker autopilot, so that when we perform an autopilot job, an experiment is automatically created with trials for the different combinations of components, hyperparameters, and artifacts. Both in training jobs and autopilot, we just have to add an extra parameter to the Python code in the SDK to define the experiment. The end result is a collection of experiments, updated in real time, that we can compare and evaluate, visually and programmatically, to choose the best model so far. Please note that while experiments permit us to keep a version of our models, we need to take care of data and code versioning ourselves. SageMaker notebooks support Git, and we can develop or integrate data versioning systems in our workflow. We can repeat the same process for the final training; since we can use the Python SDK, we can automate it programmatically. SageMaker supports additional model validation, with offline or live data, for A/B testing.

Deployment

Once the selected model is trained, we can deploy it on SageMaker in two different ways. We can set up a persistent endpoint to get one prediction at a time, using SageMaker hosting services, or we can get predictions for an entire dataset, using SageMaker batch transform. To use the hosting services, we need to create an HTTPS endpoint and pass it to SageMaker. This will allow it to be used for predictions. Note that these endpoints scoped on individual AWS accounts are not public. It is possible to use AWS Lambda and the Amazon API Gateway to allow public access, if desired (Amazon, n.d.-g).

Monitoring

Where the AWS environment really shines is in deployment, production, and monitoring support. SageMaker offers a model monitor tool to check the quality of the models in real time, while the Amazon CloudWatch model monitor enables us to set up an automated alert triggering system when there are deviations in the model quality, such as data drift and anomalies. Amazon CloudWatch Logs collect log files and notify when the quality of the model hits certain specified thresholds. AWS CloudTrail stores the log files to an Amazon S3 bucket (Amazon, n.d.-h). It is possible to detect model deviations and other potential problems early and proactively, countering the problem of model drift. SageMaker also offers functionalities to test multiple models or model versions behind the same endpoint using production variants. It is possible to test them by specifying traffic distribution, invoking specific variants, and running A/B tests. We can then evaluate the model performance, and increase the traffic reaching the best model. If necessary, we can also trigger retraining of the models, using the same capabilities for the initial training and selection.

Concluding Remarks

As we have seen in the example of Amazon SageMaker, cloud providers of machine learning as a service allow us to perform the complete model development and production pipeline, in some cases as we would use them on premises resources, in other cases, supporting us with several solved problems. Cloud providers are aimed at scalable production environments, so they offer out-of-the-box functionality like monitoring and scalability. They allow us to start small, with a small team and few resources, and grow as needed. There are tradeoffs: Even if the cloud providers are often based on open-source software (Jupyter, SparkML, Python, PyTorch, Docker, Kubernetes, Git, to name a few), the providers either modify it, or package it quite tightly with closed-source solutions. While this is not a problem per se, it impacts portability. As an example (in case we want to move our infrastructure away from SageMaker), if we have used built-in algorithms to train a model (the most straightforward solution), we would then have to reimplement the solution, or rewrite some of the code. Cloud providers also present an additional cost. The management of the infrastructure comes with an additional price tag, for example, a SageMaker EC2 instance on AWS is 40% more expensive than a bare bone AWS EC2 instance. This tradeoff is generally worth it in small organizations that do not already have IT support and the necessary resources, but if the latter are present, the cost would just be overhead. Finally, our solutions end up being developed with a specific architecture in mind, adapting to the assumptions and tools of the provider. This can be for the best, in case we are not experienced enough, since the architectures of the cloud providers are based on best practices, but it comes at the cost of flexibility. This all contributes to **vendor lock-in**. Our machine learning solution ends up linked to a provider, with potentially high costs to change (which increase as the complexity of our machine learning system increases). In general, as an organization grows, it gets to a point where it makes sense to take care of its own infrastructure. Doing this too early imposes unnecessary costs, while doing it too late presents both costs and risks. To summarize, the benefits of machine learning cloud solutions are

Vendor lock-in
It makes a customer dependent on a vendor for products and services, unable to use another vendor without substantial switching costs.

- scalable production environments,
- integrated monitoring,
- implicit best practices, and
- that it solves several of the problems for us.

Alternately, the issues of machine learning cloud solutions can be

- vendor lock-in and portability issues,
- increased costs, and
- closed source solution.

SUMMARY

This unit introduced the characteristics of the development process of a machine learning model: exploratory, iterative, and non-deterministic. We explored some challenges related to data quality and exploration, and to the development, testing, and deployment of machine learning models. We then approached the many challenges in bringing and keeping a machine learning system in production—some similar to the ones encountered in the traditional software development process, some slightly different, and some intrinsic to the non-deterministic and data-driven nature of machine learning. We explored how versioning and testing is different in the field of machine learning, and how to adapt to potential issues. This unit introduced the concept of MLOps, the practice of collaboration between data scientists and other members of the production team, and several modern tools offering solutions to some of the challenges. We finally reviewed machine learning cloud solutions offering end-to-end approaches, as well as their strengths and their weaknesses.