Managing Various Data Science Projects,
https://www.youtube.com/watch?v=qEOB0zFQRLM

UNIT II

MDSS Session 1

Managing Various Data Science Projects

Objective

Once your team start to build the project, you may realize that it requires effective management strategies.

- Can you use traditional s/w development methodologies for DS projects?
- What caveats do you have to look for?
- What processes should you use to manage development iterations?
- How can you sustain a balance between research and implementation?
- How should you deal with situations of extreme uncertainty?

This Unit II contains the following chapters:

- Chapter 7: ***Managing Innovation***
- Chapter 8: ***Managing Data Science Projects***

Chapter 7 : Managing Innovation

For most companies, DS and ML belong in the area of innovation.

Unlike s/w development, these disciplines are **terra incognita (unknown territory)**

For your customers and business stakeholders.

If you approach data science projects like any other project, you will face many unexpected problems. The domain of data science needs to be handled differently. When we step into the area of innovations, the definitions of best and good change.

We will explore innovation management and present answers to the following questions:

- Understanding innovations and exploring its management
- ***Why do big organizations fail so often?***
- Balancing sales, marketing, team leadership, and technology
- Managing innovations in a **big & start-up** company (ies)
- **Finding project ideas**

Understanding innovations

Dictionary definition of innovation: states that it is something new or something that's introduced in a different manner; an introduction of new things or methods.

Every innovation is temporary. Edison's light bulb was once innovatory, as were cars.

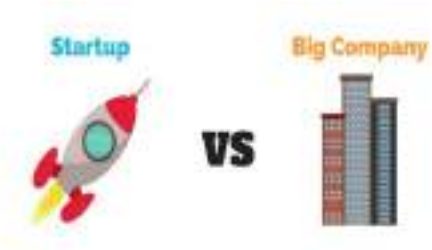
Successful innovations invade our daily lives like a tempest, change them forever, and then become mundane. Failed innovations fade and become forgotten.

If we look at history, the creation of technology does not guarantee success. Automobiles first appeared in 1886, but cars only really started changing our lives 27 years later, when the Ford Motor Company built the first moving assembly line in 1913. If you define innovation in terms of an inventor's success, the invention of the first car may be a huge breakthrough. But from a business perspective, we may only consider an innovation successful if it creates a new market with a stable income flow. Technical innovation advances the field, while business innovation creates some new, previously unavailable value. This value helps new markets emerge and new businesses rise.

Why do big organizations fail so often?

Big companies often see innovations as a gold mine. Many entrepreneurs think, Have you heard about this new artificial intelligence start-up? Just imagine what we can achieve in this field. Our resources are vast compared to other companies. However, history dictates otherwise, since

emerging innovative technologies are often born inside small start-ups rather than in big, stable businesses. This is counterintuitive. Large companies have more resources, people, time, and risk immunity, while start-ups have almost none. Then why do big corporations fail at innovation?



Why do big organizations fail so often?

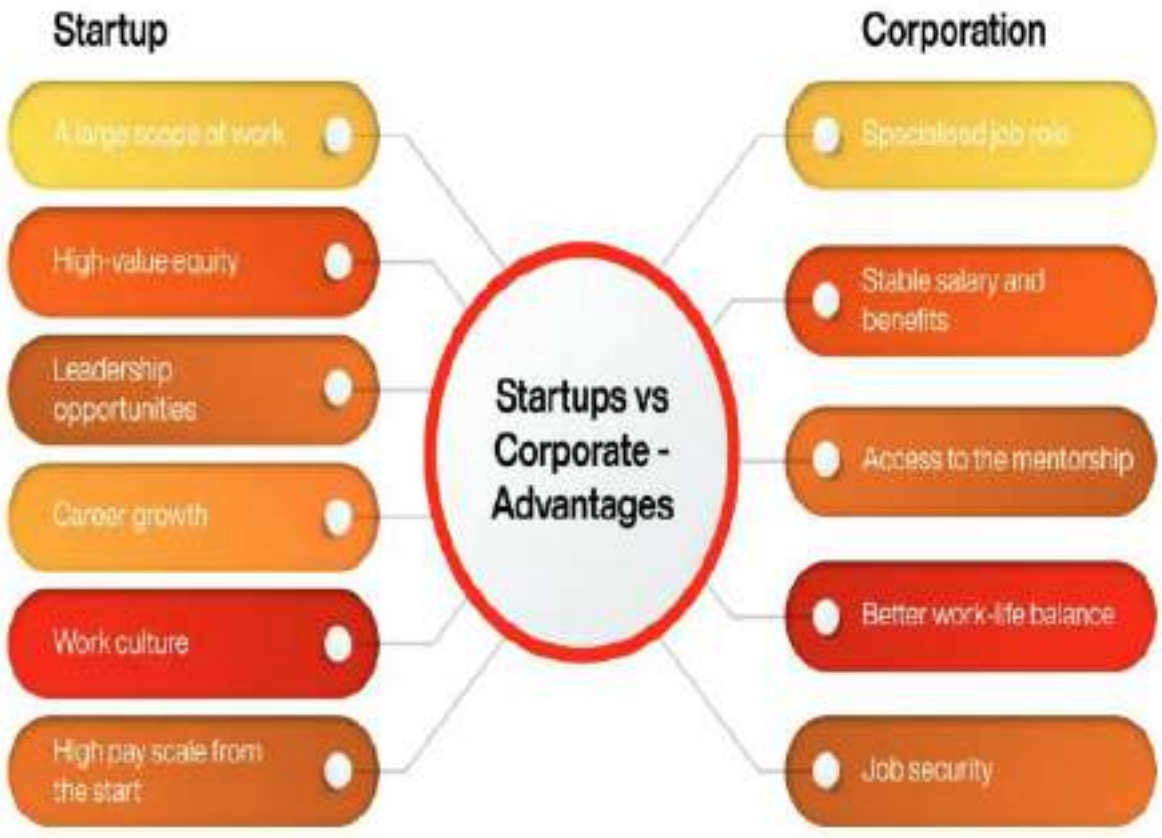
Innovations generate little to no revenue in the first stages of development. In large companies, innovative products will often compete with the best offers of the company. If you look at this from this angle, innovation seems legitimate in a start-up company, while being incompatible with the existing revenue channels of a big company. Small companies strive to get income, and competing with large corporations in developed markets is not an option. Small start-ups won't have enough resources to deliver competitive products that the market needs. Thus, the natural way for start-up survival is innovation.

New technologies are raw and unpolished, so they seem unattractive to new markets.

Case of Hard Disk Drives **(HDDs)** as an example of this.

14 “ , 8 “ , 5.25” 3.5 “ and now……

Starting innovative projects in large companies is easy, but finishing them is troublesome. Innovations appear as nonsense according to a company's short to mid-term goals since they bring in no revenue and require large investments. However, bringing innovations to large organizations is not impossible. We will explore how we can grow ideas into the businesses of small and large organizations as we progress.



Game of Markets

DS is still a young field with great potential. Many experts state that even if we were to suspend new research, if we were to integrate new technologies that have emerged in the last 10 years, we will still be two to three decades ahead of the game. However, there is a large gap between research projects and widely applicable technologies. Like HDDs, a research project needs to grow its features in order to outrun its current competitors.

DS uses research in ML, DL, and statistics to deliver software products that solve real-world problems. To become usable and widely applicable, software needs large time investments. The time of software engineers, data scientists, and other technical experts' costs money.

End of session 1

Game of Markets,
https://www.youtube.com/watch?v=gAchz6S_Ckc

UNIT II

MDSS Session 2

Game of Markets

DS is still a young field with great potential. Many experts state that even if we were to suspend new research, if we were to integrate new technologies that have emerged in the last 10 years, we will still be two to three decades ahead of the game. However, there is a large gap between research projects and widely applicable technologies. Like HDDs, a research project needs to grow its features in order to outrun its current competitors.

DS uses research in ML, DL, and statistics to deliver software products that solve real-world problems. To become usable and widely applicable, software needs large time investments. The time of software engineers, data scientists, and other technical experts' costs money.

To earn money from your software, you need to solve people's problems. Your products need to bring something valuable to people. When this value is large enough to sustain a constant demand, you create a new market. This market is ruled by the laws of supply and demand: people and companies will exchange money for your products and services.

In their infancy, big organizations created their own markets or took part in existing ones.

Markets can age and die like we do, giving new markets somewhere to live. If company's in

a dying market do not transition to an emerging market; they cease to exist.

Creating new Markets

Creating new markets is complex and risky. If you take too long

and use too many resources to test your idea, you will fail.

The risks are high, so testing ideas should be quick and easy. This testing allows you to convert mistakes into knowledge and improve your product, iteration after iteration. *Failing fast and constantly learning by testing ideas are the key points behind the lean start-up methodology.*

The formula is simple: probe as many ideas as you can, iteratively pivot your product to match the market, and increase the demand for your services. The key characteristic of this process is speed.

Creating new Markets

Big companies rarely operate quickly. Long operational cycles and heavyweight organizational processes come up naturally with the company's scale. Without those crucial instruments, the company may fail to operate.

Nonetheless, innovations do not tolerate long cycles. The cost of failure increases with the time span of an experiment, which makes innovations extremely costly for large businesses. For organizations, data science is still an innovation, and most market niches haven't been developed or discovered yet. ***While innovations are hard, they are not impossible***.

Exploring Innovation Management

Innovations are very chaotic in nature. Innovators call for experimentation, but they cannot predict the end results and struggle to define deadlines. These qualities of innovations make them hard to implement in a business environment, where clearly defined goals, strict deadlines, and finite budgets are the norm. Innovation management provides a set of techniques that bring order into a chaotic realm of innovations.

The word *management is* associated with direct control, but this is not the case for innovations. Freedom is critical for every innovation. You won't get any positive results by trying to micromanage. Innovation management is about providing support and integrating innovations into existing businesses so that they deliver helpful results.

Different types of innovations:

1. **Disruptive innovations** are what most people understand by the word innovation. This type of innovation brings drastic changes to the market. It introduces something that's new and technologically mature enough to create a new market. The iPod, iPhone, IBM PC, and electric light bulb were all disruptive innovations.

2. Sustaining innovations feel more organic and incremental. A new version of

your favorite operating system, the UI changes of social networks, and iPhone upgrades are all sustaining innovations. The goal of sustaining innovation is to keep up with competitors by introducing new features that will keep your customers from buying into a more attractive offer from a rival company.

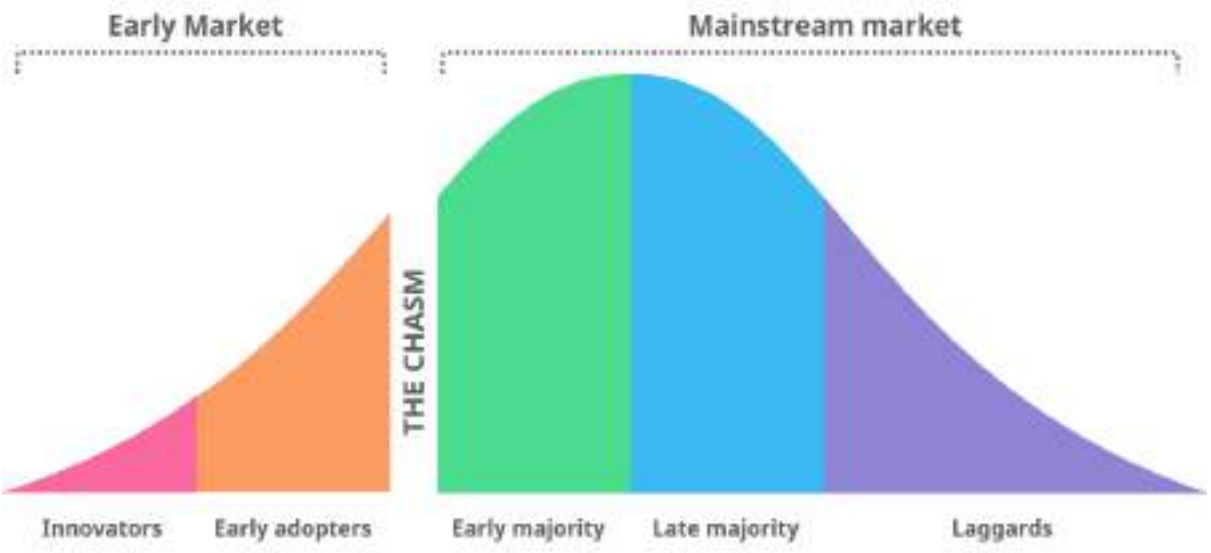
Each sustaining innovation increases the gap between competitors slightly, while disruptive innovations change the way we live and work, thus creating new markets and making old ones obsolete. From a layman observer's perspective, disruptive innovations may appear all of a sudden, but in reality, they are not a flash of insight. *The Innovator's Dilemma contains many examples of disruptive innovations that grew in small markets,* where features of innovative products were more appealing for very specific use cases. A stable source of income allows innovative technology to grow to its potential, while companies that provide old-generation products do not see the direct competition on them market. When the new product disrupts the market, it is already too late.

We can view both types through the following life cycle:

1. **Search: In this phase, the company searches for an idea to experiment with.**
2. **Build: Here, the company builds prototypes, conducts experiments, and** measures results.
3. **Scale: At this stage, the build phase gives us a promising direction to work on, so** the company scales out. Prototypes are transformed into market-ready products and start to generate income.
4. **Expand: Every innovation saturates its market. At this stage, the company** expands its product to new markets to keep the growth rate intact.
5. **Sustain: At this stage, the innovation becomes a stable business. Some companies**

can live for decades in the sustain stage. However, this comes with risks. Your current position in the market is not a constant irreducible resource of income. It can quickly saturate without long-term investments. Each minute of inactivity opens up new opportunities for your competitors to create disruptive innovations that can throw you out of the business. The sustain phase is a flag,

signalling that the company needs to resume the **Search stage.**



New technology can quickly find a seemingly stable source of income in the early market that comprises companies who are willing to pay money for risky, untested, experimental, but promising solutions.

Many organizations think that the transition from the early market to the mainstream market will be smooth; the early market will produce enough income so that it can create new features that will be valuable to the mainstream market. However, this is not the case in practice.

The gap between early and mainstream markets is much larger than it seems.

When the early market saturates, the company enters the ***chasm***. Its offering is not mature and developed enough for the mainstream market, where customers are used to feature-rich and stable products.

So, the company is forced to live with a diminished income stream, while its product requires large investments so that it can create the features that have been requested by the mainstream market.

To find a way out of the chasm, the company must focus its efforts on a very specific area of application where its product's best features are more important than its current flaws. The goal is to find a tiny but reliable niche and concentrate on perfecting it. This creates a small market that will allow you to grow momentum through success stories, functionality, customer references, and new income streams. Consistent success in a small market will give you the resources and time that's necessary so that you can scale out to more mature markets.

End of session 2

Integrating Innovations,
https://www.youtube.com/watch?v=PSy4WMKI4t8

UNIT II

MDSS Session 3

Integrating Innovations

Innovations require experimentation. New approaches fail often, but failure does not signify the end of the line. Failure just means that you have to distill the experience, learn from your mistakes, and try again as fast as you can.

If you try to do this in a developed company, their processes and best practices may make you look like you are insane and that your actions may appear destructive and contradicting the company's values and goals. This happens because most businesses are oriented toward sustaining innovations.

Existing values, goals, and KPIs react to disruptive innovations like an immune system would react to a virus—with defensive actions. This is the reason why most **R&D** departments live a hard corporate life. They pursue goals that are incompatible with the values of the company they work for. To ease the integration of disruptive innovations into a large company, you may try to introduce another isolated business entity inside the company.

There are different ways to do this:

Acquisition: Buy companies that have already managed to cross the chasm. The main challenge of this approach is to integrate the acquired company. Both technological and organizational integration needs to be done carefully so that it does not badly influence newly acquired products and teams.

Separate entity: Create a branch company that will receive financing from the head company. The main challenges of this approach are large costs, risk management, and resisting the urge to have complete control. Honestly evaluate your company's potential in order to create and organize a new business with a set of goals that are aimed at the long-term development of disruptive technology.

Separate department: This is the hardest way to create the innovations team. The main challenge here is creating a new set of goals and values for the existing company.

Thinking about data science projects from the perspective of innovations is helpful in many ways. Machine learning and deep learning can create sustaining and disruptive innovations. Innovation management will give you a broader context and let your main strategy succeed in the market. Organizations that do not integrate innovations will be outrun by their competitors. It is easy to blame technology for not being mature enough, or experts for not being as good as the competitor's rockstars. While doing this, many companies forget to consider the validity of their strategy for forming markets around new technologies. Data science solutions need a stable organizational foundation and a plan of action for income generation. Innovation management helps us take care of both components.

Balancing sales, marketing, team leadership, and technology

To thrive in data science management, you need to find a balance between different specialties. The data science manager switches between the tasks of sales, marketing, and optimization every day. But aren't they supposed to care about data science the most? Since we do our jobs collectively, we tend to communicate a lot. Ask any technical expert working in a business environment about how much time they spend doing actual work.

On average, a software engineer will tell you that they spend 2 to 3 hours coding. During the other 6 hours, they attend meetings, write or read documentation, create tickets, and discuss technical designs. Data scientists spend a lot of time talking about data definitions, metric choices, and the business impact of the model they are building.

The number of areas a data scientist can spend their time on is bewildering.

Managing Innovations in a Big Company

Most companies already know about the transformative power of data science. No one wants to fall out of competition, so companies create internal data science and R&D departments. Working in this environment is challenging because you are seen as the driving force of innovation by some and as a disturbing annoyance by others. Innovations often lead to significant changes in the existing business processes of the company, and many will not want to see these changes implemented.

Bringing disruptive innovations to existing businesses is a tricky process; to increase your chances of success, make sure you have the following:

1) **Organizational power:** You will need direct or indirect organizational power to implement changes.

2) **A migration plan:** You will need to have a complete vision of as-is (before integration) and to-be (after integration) business processes. You will also need a migration plan that guides you from as-is to to-be processes.

3) **Success criteria and an integration strategy: Making an abrupt change to the** entire business process and quickly integrating innovations is a risky strategy. You should have measurable success criteria for your project and a risk management strategy that allows you to take control of losses in the worst-case scenario.

4) **Open processes and clear explanations:** Everybody that's affected by the migration will need a crystal clear understanding of what is happening and how it will affect their workflow.

5) **Involvement in integration:** You will need help in describing as-is processes and help with the migration plans. No one will be better at this than the people who already execute the as-is business process. Find people who are involved in the day-to-day execution of the existing process and make sure that they participate in the processes mentioned in points 3 and 4.

Managing innovations in a start-up company

If you work in a start-up, innovations will feel more natural and easier to implement than in a large, slowly moving business. However, there is a significant difference between a start-up and an established company. Start-ups cannot afford to make a lot of mistakes. Stable income streams allow large companies to test many ideas simultaneously, whereas start-ups are often limited to evolving in one simultaneous direction.

The concept of the chasm is important for start-ups. The main goal of a start-up is to find a new market where it can grow upon itself and provide value that doesn't exist. Matching your idea and your technology to market demands involves a lot of quantifiable experimentation. It is important to use a measurable approach that is advocated in *The Lean Startup and Crossing the Chasm methods.*

Finding project ideas

Before implementing your first prototype, you will need to find promising ideas to work on. To create ideas, you can work from one of two starting points: business or data.

1. Finding ideas in business processes

If you are working with an already established business, the first and the most obvious way to find ideas is to ask management about their needs. Merging DS expertise with deep domain knowledge will allow you to find a match between their requests and your team's capabilities. However, your customers will rarely have a problem they know about.

The most likely response you will get while using this strategy is, *We know of this problem, and we are already working on the solution.*

On rare occasions, a problem may be presented to you on a silver platter. If that's not the case, you will need to dive deeper into the company's business. Start with their key revenue-generating processes. Educate yourself about how they work. Then, mark all the steps as manual, partially automated, or fully automated. Two key areas for the improvement of an existing business process are the automation of manual steps and the improvement of automated steps.

The best course of action when analyzing existing business processes is to find a step that is as follows:

Generates data: This or any previous step of the process should constantly generate digital data that can be used by a model.

Is business-critical: The improvement should provide direct business value to your customer. Increasing the value is always a strong selling point. **Has potential applications for data science:**

You can use data to build an algorithm that provides insights or automates decision-making processes

2. Finding ideas in Data

Another way to create project ideas is to look directly into the data:

1. First, you should create a data source map of the company you are working with. Sometimes, this information will be readily available for you to use. If it isn't, then your team should do some research and recover the data architecture of an organization. Using this data map, you can identify the largest and most complete data sources.

2. Then, sort all the data sources by the amount of data and data quality and look over possible data merges between different databases. The goal is to create several starting points for further research.

3. Finally, look deeper into the data sources you have selected. Seek datasets that can be used to build supervised or unsupervised models. At this stage, you should not build any models; just write down all your ideas—the more the better. After jotting them down, see if any of your ideas can create new business value.

This way of finding ideas is risky, time-consuming, and very creative. The chances of you succeeding are lower than in the previous approach, but if you are able to find a working idea that works bottom-up from the data, you are much more likely to stumble upon a unique data science use case that gives you a competitive advantage and provides new business value. The first approach is much more suited toward finding sustainable innovations, while the second approach can be used to mine for disruptive ideas, that is, transformative ideas.

Once in a while, some people come up with truly unique ideas. There may not be any data, nor a business—just a captivating idea. Transformative ideas that disrupt the industry often

fuse several disciplines that have no close relationship at first glance:

Medicine + computer vision and machine learning = automated disease risk analysis for patients

Automobiles + computer vision = self-driving cars

Call centers + deep neural networks = fully automated dialogue systems with speech recognition and generation

If you think that you have come up with a promising concept, do not rush it. Write down your idea and start researching it:

Look for similar products that are already on the market. Chances are, you will find several at least vaguely similar products.

Collect and classify a competitor's product features. Write down the core business model behind each product. Rank all the collected features into several categories:

Best feature: No one does this better; it's a distinctive feature of the product. This feature is often the defining purchase factor.

Core feature: The majority of competitors have this feature implemented fairly well.

Basic feature: This is implemented in a basic way. Lots of other competitors have a more complete implementation of this feature.

Lacking feature: The product lacks a feature.

Start fleshing out your idea by describing the product's features. Create a profile for your product and think of the feature that distinguishes it from others. Think about your product's business model. How would it relate to other products on the market?

If you can't find any competitors on the market, or you can see that your product could easily throw competitors out of the market, then you have a chance of creating a disruptive technology. Remember that pushing disruptive innovations into the market requires careful planning, focus, and a strategy for crossing the chasm.

End of session 3

Understanding DS Project Failures,
https://www.youtube.com/watch?v=BUIAxnCqRS8

UNIT II

MDSS Session 4

Managing Various Data Science Projects

Objective

Previously, we looked at innovation management. We developed recipes that can help find ideas for data science projects and matched them with their market demand.

Now we will cover the non-technical side of this by looking at how DS projects stand out from general software development projects.

We'll look at common reasons for their failure and develop an approach that will lower the risks of data science projects and also dive into the art and science of project estimates.

Topics:

- **Understanding DS project failure**
- **Exploring DS project life cycle**
- **Choosing a project management methodology**
- **Choosing a methodology that suits your project**
- **Estimating DS projects**
- **Discovering the goals of estimation process**

Understanding DS Project Failure

Every DS project ends up being a software system that generates scheduled reports or operates online. The world of software engineering already provides us with a multitude of software project management methodologies, so why do we need to reinvent a special approach for data science projects?

Answer: DS projects require much more experimentation and have to tolerate far more failures than software engineering projects.

Look at the common causes of failure for DS projects:

Dependence on data: A robust **customer relationship management** (**CRM**) system that organizes the sales process will work well in many organizations, independent of their business. A system that predicts the outcome of a sales process may work well in one organization, but will require a partial rewrite for another organization and may not work at all in another. The reason for this is that machine learning algorithms depend on data, and every organization will have its own data model of its customers and its own sales process.

Changing requirements: While software development projects often suffer from changing requirements, the changes mostly flow from the customer to the implementation team. In data science projects, new insights and research results from the implementation team can create a feedback loop. Project stakeholders can generate new requirements and change the course of the project based on the new information that's discovered by data scientists.

Changing Data: In software development projects, the data model is mostly fixed or can be changed in a controlled manner. DS projects often need to be integrated with new data sources for research purposes. Data is always changing and transforming, creating multiple intermediate representations inside the system. People and software components use these representations for reporting, data processing, and modeling. Software engineering projects use fixed or slowly changing data models, while data science projects use constantly evolving data pipelines.

Experimentation and Research: Data science projects involve completing many experiments. Typically, the number ranges from hundreds to thousands.

Software engineering projects limit research by designing a system architecture and evolving it in a controlled manner. In data science projects, the next

experiment may turn the project in a new direction, and you never know when this will happen.

Understanding DS Management Approaches

The traditional management approach to software engineering projects was not built with these problems in mind. The key problem that most modern software project management methodologies need to solve is the issue of changing requirements.

Agile methodologies focus on planning and executing fast iterations. Each iteration aims to deliver functionality to the client as fast as possible. External feedback is the primary source of changes in the project.

In DS projects, changes come from every direction. They spread internally from the project's team and externally from the business' customers.

Metrics should always confirm progress. Getting one step closer to your goal may take tens or even hundreds of failed experiments, which makes fast iterations a must.

The typical iteration length of an Agile project can stretch from 2 weeks to 1 month. The

project team fixes the iteration scope for this duration and delivers it under a strict timeline.

In a DS project, an experiment's result in the middle of the sprint can affect the sprint's goals and make working on other planned tasks less important due to the new

discovery.

Management must provide a safety net for common issues and problems. Methodologies

that come from the SE domain can give you a solid foundation here, but they do not provide any tools that we can use to manage research and govern data.

If you develop systems that use machine learning under the hood, it is necessary to take

care of the following:

Requirements for validation and alignment: You need to detect and manage requirement changes from external (customers) and internal (research team) sources.

Data governance: Your project will need data governance standards, which should be rigorously applied to each piece of code that works with data. Ideally, each row of data going through your

pipeline should be tracked back to its data source. All incoming and outgoing datasets, including intermediate reports, should be tracked and documented.

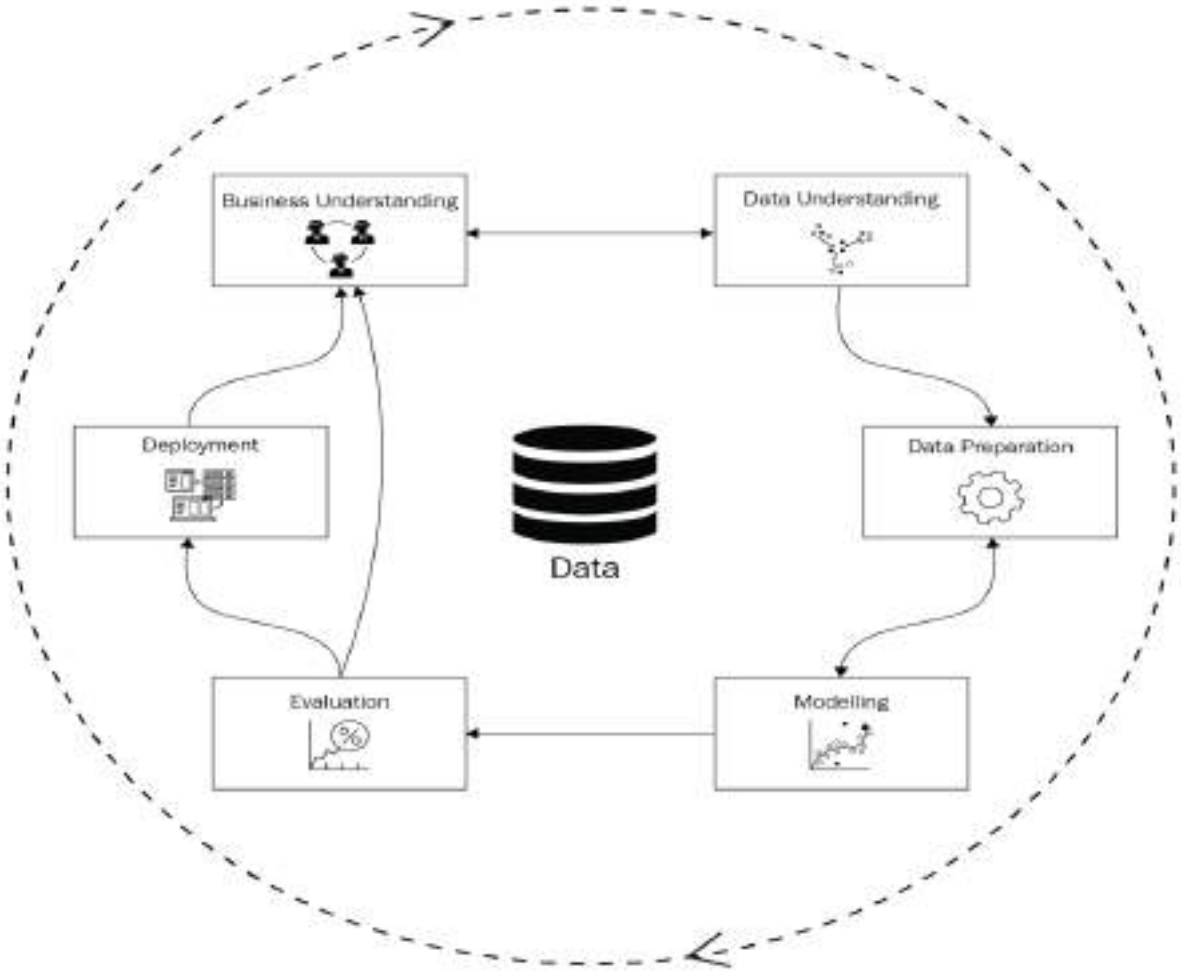
Research processes: Each DS project will need to be researched extensively. Without control, research can quickly eat away at your budget without project completion being in sight. The essential components for managing a research project include the following:

- **Research planning:** The project team should plan and prioritize all **of** their research.
- **Experimentation methodology:** Each experiment should conform to a set of standards such as tracking, documentation, and reproducibility.
- **Fail fast and recover early:** Experiments often fail. Your management approach should make experiments fast so that your team can iterate and learn as quickly as possible.

Software engineering processes: Much of your work will be in creating software. Software project management already offers great tools for this, but they need to be tightly integrated with all the other components of the management methodology.

Exploring the data science project life cycle

Each data science project has several distinct states. We can structure projects in different domains and different technologies into stages that comprise the data science project life cycle, as shown in the following diagram:



Business Understanding

During this stage, you apply your domain expertise and research the business side of the project. You define the business requirements and confirm that their implementation would make the lives of your customers better. You should also define and document all the relevant business metrics that will allow you to measure and report on results in a way that is understandable by the business side. The output of this stage should be a business requirements specification that has been viewed, redacted, and agreed upon by the project stakeholders.

Data Understanding

During this stage, you research the data architecture of the organization you are working with. You document data sources, their owners, and the technologies they use.

You should not document every available data source unless you want to mine project ideas from data.

Focus on data that's useful for the project. After finding this data, perform an **exploratory data analysis (EDA**) and research the data thoroughly. Look at any anomalies and unusual artifacts in the data. Study the reasons behind their occurrence and document ways to handle them.

For example, if the dataset has a lot of empty values, you should have a plan for how to deal with them, and your actions should not distort the data in an undesirable way.

You should also look at ideas regarding feature engineering during the EDA stage. Perform statistical analysis on the data and try to find causal relationships that will help to solve the task at hand.

The data understanding stage should have the following outputs:

- **Data source dictionary: This document briefly describes all the data sources that** are relevant to your project.
- **An EDA report that shows the conclusions of your data analysis: This** document should describe the approach that you will use to solve the task at hand and the strategies for handling errors that you have found in the data. You should include facts that may interest your customer.

Data preparation

This stage is where we start working with data. The data preparation stage involves taking raw data and changing it into a useful format. You read data from its sources and prepare it

so that you can use the data to reach the project's goal. If you are solving a task based on structured data and plan to use machine learning, you will need to perform feature engineering. The previous stage should give you insights into the quirks of the data that you can fix during the data preparation stage. This stage's output is one or more reproducible data preparation.

Optimizing Data Preparation

Data preparation and data understanding are surprisingly time-consuming. These stages can take up to 80% of the project's time, so don't forget to plan in advance. Since this stage is time-consuming, optimizing the team's performance is important. Open source tools for automated EDA and feature engineering can save you a lot of time at the start of the project, so don't hesitate to use them. In the *Creating the Development Infrastructure sessions,* we will look at several libraries that you can use to speed up the data preparation and data understanding stages.

Modeling

This topic was covered in the *What is Data Science? section of this book. In this stage, we* apply our knowledge of data science, machine learning, and deep learning to solve the task at hand. This is done in the following stages:

1. First, we determine the task type, that is, supervised (classification and regression), unsupervised (clustering and document topic modeling), or reinforcement learning.
2. Then, prepare a list of algorithms that are suitable for solving the task.
3. Next, come up with a model validation and testing approach.
4. Finally, optimize the parameters of the model and select the best model.

Evaluation

While not being separate from the modeling and deployment steps, this stage deserves to stand on its own. You must test technical and business metrics, as well as checking the individual predictions of the model at this stage. Look at the biggest errors the model made on the test set and think about the changes that you can make to your data, features, or models that can fix those errors. This is also a good way to spot data processing bugs.

Your project should have two evaluation strategies: online and offline. Online evaluation takes care of tracking all the metrics for the already deployed model, while offline evaluation is used to decide which model will make it to the deployment stage.

Typical data science projects contain hundreds of experiments with different models, data, and parameters. Each experiment generates new data in the form of metrics, code parameters, and notes. Use a specialized experiment tracking tool to decide on the success or failure of a particular experiment. These tools can automatically collect all the logs, metrics, and artifacts of the experiment to ensure their reproducibility and to ease searching the experiment results. If you don't want to or can't use a special tool, a spreadsheet can be a good substitute, although you will need to spend more time working on it. Having a complete log of all the experiments and decisions you've made regarding modelling and data pre-processing will help you compare different experiments and make conclusions about their results.

Deployment

At the deployment stage, you publish your best model for your end users and examine the results. At this stage, complexities are often overlooked. Production code has a separate set of strict requirements and **service level agreements (SLAs) that your model needs to meet.**

We can separate those requirements into two categories: functional and non-functional. Functional requirements define your service's features, while non-functional requirements define your SLAs. Some examples of the functional requirements for your model service are as follows:

- Request/response format
- Capability for model versioning
- A UI for tracking deployments and request statistics

Non-functional requirements define the quality of service and availability of your service, and some of them are as follows:

- Desired request throughput (1,000 requests per second)
- Availability schedule (24/7 and 5/8)
- Secure communication
- Elastic scalability so that the system will stay available when the user load peaks

Deployment

At the deployment stage, you publish your best model for your end users and examine the results. At this stage, complexities are often overlooked. Production code has a separate set of strict requirements and **service level agreements (SLAs) that your model needs to meet.**

We can separate those requirements into two categories: functional and nonfunctional. Functional requirements define your service's features, while nonfunctional requirements define your SLAs. Some examples of the functional requirements for your model service are as follows:

The requirements for model deployment are similar for different projects, so this part of the process is subject to reusability. Instead of repeating the same work for each project, you can develop your own model-serving framework or use an existing one.

Another important point to remember at the deployment stage is evaluation. This does not end at the previous stage; you should evaluate all of the model's metrics online. Your system may trigger alerts or compensative actions such as model retraining if the online metrics drop below a certain threshold. A/B testing and multi-armed bandits are also a part of the deployment process and can be supported as features of your model server.

Now, you should be familiar with the common stages of each data science project.

End of session 1

choosing a project management methodology, https://www.youtube.com/watch?v=CwbFFB6-5Sk

UNIT II

MDSS Session 5

Choosing a Project Management Methodology

PM methodologies provide a set of rules and processes that can distinguish chaotic projects from coherent ones. They provide a framework where everyone can act toward a greater goal. Laws do the same for our society. However, laws are not perfect and they often fail. There is no silver bullet in the world of software management either. Some management practices are better suited to one type of project and will let you down in another. In the following sections, we will explore the most popular ways of managing software projects and learn how to adapt them to a data science environment so that we can draw conclusions and choose the one that suits our project the best.

Waterfall

The most intuitive way to manage a project is to approach it like you're building a house.

The steps for this are:

	House	Software
1	Prepare the building site	Prepare the development environment
2	Lay a foundation	Analyze and document the requirements
3	Create a framework	Analyze and document the architecture and software specification
4	Build a roof	Build the system
5	Connect the electricity and water	Test that everything is working according to the requirements
6	Finish the exterior and interior	Finish the project

Reasons for not suitability:

All steps are laid out sequentially and are only repeated once. If you make a single mistake, the project plan will fall apart. A single undocumented requirement, such as the one at *step 2*, can result in a disaster at *step 6*. Clients do not have a complete view of the end result and they can make mistakes too. Requirements can change after customers see the actual implementation of their requests.

Addressing the Issues:

Software project managers know that a single waterfall won't solve their issues, so they compose many smaller waterfalls into sequential iterations. This stage of evolution is called iterative and incremental software development. The iterative project is comprised of several phases that are managed in a waterfall fashion. The length of a single iteration is measured in months. At the end of each phase, the development team shows intermediate results to the end user for the purpose of collecting feedback. This feedback is used to jumpstart the next iteration. With each cycle, the understanding of the desired result evolves until it satisfies the customer's needs.

Agile

The iterative approach is still too heavy for most software projects. They suffer from changes that pile up in a mountain of technical requirements. In 2001, some of the brightest heads of the software development world created an Agile manifesto, which described a new management approach in four simple points:

1. Individuals and interactions take precedence over processes and tools
2. Working software takes precedence over comprehensive documentation
3. Customer collaboration takes precedence over contract negotiation
4. Responding to change takes precedence over following a plan

That is, while there is value in the latter of each point, we value the former of each point more.

Kanban

The best metaphor for explaining Kanban is a conveyor belt.

Imagine that all of your tasks go through a fixed number of stages before they're finished. The concrete definition of those stages is up to you. In software projects, you may want to use the following process:

1. Backlog (a buffer where all incoming tasks are collected before

they're processed)

2. Requirements specification

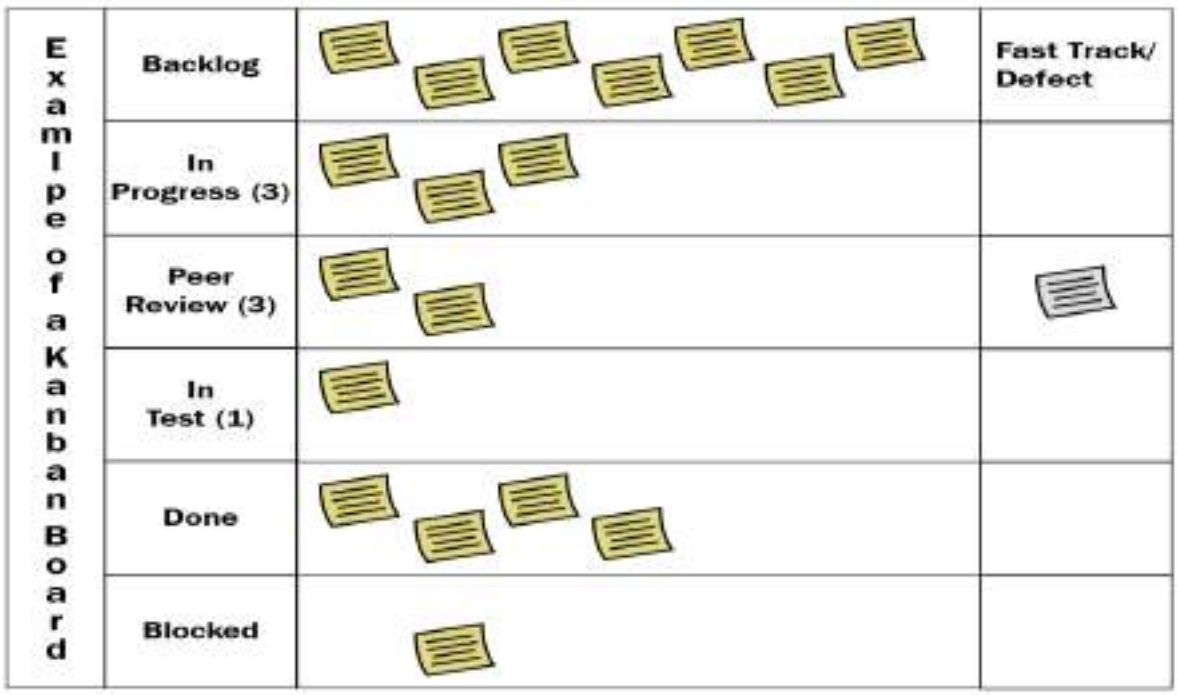
3. Development

4. Code review

5. Testing

6. Deployment

Kanban visualizes each task on a board, as shown in the following diagram



Each stage should have a limit regarding the tasks that can be done in parallel. Kanban purposefully limits the number of simultaneous tasks to increase throughput. If the team becomes blocked

because there are too many tasks sitting in the deployment stage, then all the team members who are capable of making shipments of the end product to production

should stop doing their tasks and switch to getting rid of the bottleneck. Once the problem has been resolved, the team may decide to work on other tasks according to their priority.

Because of this, Kanban favors cross-functional teams where everyone can help push tasks through each stage. Kanban does not remove the concept of roles, but it states that no matter the role, each team member should be able to help in dealing with a bottleneck.

Kanban focuses on completing a single task from start to finish as fast as possible. The main metrics that we can use to measure the effectiveness of Kanban projects are as follows:

Lead time: The time it takes for a task to move from the backlog to being completed on average.

Cycle tile: The time it takes for a task to move from the starting stage to being completed on average. In our example, the cycle time would be between the requirements specification and deployment.

Throughput: The average number of tasks you can get done during a time interval, that is, a day, a week, or a month.

In general, you don't create a fixed project plan with fixed deadlines when using Kanban. Also, you don't have to estimate each task individually since the metrics will take care of that. Measure your team's throughput for several weeks so that you have an idea of how

much you will be able to deliver in the near future.

<u>Kanban's powers are also its limitations, some of which are:</u>

- Kanban works best when the amount of work in each of your tasks is the same. If some tasks are taking significantly longer than the others, your metrics will stop being useful.

- If you don't want to work as a cross-functional team, your throughput will suffer from bottlenecks, which makes using Kanban worthless.
- Kanban does not give you the tools you need to manage deadlines, project scope, and budgets. The single thing it takes care of is optimizing throughput.

Kanban is a great s/w management approach for projects with repetitive tasks. It can also be partially applied to parts of your project where it makes sense to use it. Here are some examples of projects where using Kanban is a good idea:

- In a s/w support project, where you should take care of deployment and fixing frequent issues.
- If your DS project has a dedicated team that performs experiments with ML models, using Kanban will help increase their throughput.
- Projects where you need to create lots of similar things, such as hundreds of web forms, data mappings, or identical machine learning models.

Scrum

Another popular management methodology from the agile family is Scrum. The main idea behind Scrum is the sprint. The sprint is a set of tasks with a fixed deadline and duration. Typical sprint durations are one week, two weeks, and one month.

The Scrum process steps:

1. Backlog grooming

2. Sprint planning

3. Sprint execution

4. Retrospective

Akin to other agile methodologies, all the tasks go into the project backlog. The project backlog needs periodic grooming: all of the obsolete tasks should be deleted; the rest of the tasks need to be ordered by priority.

The main component of Scrum is the sprint. A sprint is an iteration with a fixed deadline and a defined goal. The typical length of a sprint is 2 weeks. It always starts with a sprint planning meeting, where team members observe the project backlog and take tasks into a sprint. Each task is estimated in abstract story points. The goal of estimating in story points

rather than ours is to make estimations relative rather than absolute.

Example: Consider a task with one story point trivial and a task with two story points as being slightly harder but still easy to complete. Four to six story points would indicate a normal task. Another system of story point estimates suggests using powers of 2 to 128 as task estimates.

On the first sprint, the estimations are fairly approximate. On the second sprint, you compare the new tasks with the previous ones and see how many story points the task is worth. After four sprints, you can see how many story points your team can complete on average. You can also calculate an approximate hour equivalent of a single story point, although this should only be used as a reference and not as a substitute for story points

during the sprint planning process.

The sprint starts after the planning phase. When you start working on the sprint, it will be locked. You cannot change the scope you defined during the planning phase. The primary focus of the team is to complete all the tasks in the sprint before it ends. This strategy allows you to achieve your planned goals while being robust to changes. The main strength of Scrum is also its main weakness. If your customer comes into work in the middle of the week with an *extremely important task that needs to be done ASAP*, you should do your best to convince them that the team will deliver this during the next sprint. Scope locking is an essential mechanism that makes sprints work. If you depart from this rule often, Scrum will become an obstacle rather than a beneficial and effective management approach.

In practice, scope locking can cause problems, especially if you are working in a B2B environment. For situations where you have no options and are forced to change sprint scope, you have two options:

Trade tasks: You can remove a task from the sprint and add a new one.

Start a new sprint: You can stop the current sprint and plan a new one.

Using these options frequently makes Scrum ineffective. Try to negotiate a fixed sprint scope with your customers and show them that it brings benefits such as planned delivery while leaving space for requirement changes.

Choosing a methodology that suits your project

Choosing a project management methodology can become a captivating and complex task. You can spend a lot of time thinking about how one approach will support your processes better than another, and what limitations it will have. Try not to spend much of your time on methodological considerations. It is much more important to choose something and stick with it unless it is clearly harming your project. To simplify this process, we will explore some simple guidelines when it comes to choosing a management approach.

Creating Disruptive Innovation

If you create a solution that should disrupt the market, the only thing that matters is the efficiency of your methodology. You won't have many customers at the start of the project, so you should be able to collect feedback and perform focused work to iterate on the next version of your product. Scrum works best in such situations. You can implement new features regarding the sprint and collect feedback at the end of each sprint to start a new iteration. Kanban will work too, but it will provide fewer benefits in terms of disruptive innovation.

Providing a tested solution

If you implement a system that resembles some of your past projects, it will presumably require much less research than in previous iterations. This is also the case for system integration projects, where you provide services that can integrate your product into the customer's IT environment. In those projects, you can define many customer-focused tasks that can be divided into three to five groups depending on the total amount of work that needs to be done. In this setting, Kanban will provide the most benefit. Using Kanban will allow you to focus on delivering more results to the customer in less time.

Developing a custom project for a customer

Using Agile methodologies can be extremely tricky when you're working on a project for a customer. Your clients will want to have the best of both worlds: fixed deadlines with constantly changing requirements. Your job is to decide on the best approach for this project and explain its pros and cons. Many teams settle for something between Scrum and waterfall: you develop the initial scope of the project, estimate it, and show it to the client. Next, you implement the project scope piece by piece using sprints. The requirements will inevitably change during the implementation stage, so it is important that you manage these changes and keep the customer involved in sprint planning.

Choosing a project methodology goes hand in hand with estimating data science projects. In the next section, we will define the goals of the estimation process and learn how to make estimates.

End of session 5

Estimating DS projects ,
https://www.youtube.com/watch?v=IN-X93KOFyM

UNIT II

MDSS Session 6

Estimating DS Projects

If you need to explain the basic principles of forecasting to someone, ask them if they have ever worked on a software project. If so, they already know the basics of forecasting: everyone who has worked on one has estimated tasks. Everyone needs estimates. Your customers need them to plan and control when they will start to use the results of your project. The project manager needs estimates to understand the scope, amount of work, and approximate costs for individual tasks or an entire project.

Estimation is beneficial in several areas, such as the following:

Understanding work structure: Break down a task into multiple subtasks to view the main steps that you need to complete.

Understanding complexity: While it is hard to estimate a complex task by itself**,** estimating each individual part of the work structure is simpler. It allows you to get an idea of how complex the task is and how long it will take to finish it.

Understanding costs: In most businesses, you won't be able to start working on a project if you don't explain and defend its costs and required resources first. The largest problem with estimations is that they fail. Our plans are inaccurate, incomplete, and often irrelevant to how the real work will be done. Even experienced software developers struggle to estimate the total amount of hours that it will take to do a single task unless they have done it multiple times already.

Research shows that humans are bad at absolute estimates. Our brains are simply not suited to making accurate mental models of complex multi-layered projects. For example, if we were to ask a bunch of strangers about the height of the nearest building, the majority would fail to give you the correct answer. However, if you told them the height of several buildings, their estimates would be much more accurate. This is true not only for building

height estimates but for all kinds of estimation.

To use relative estimates in data science projects, you need two things: relevant samples and a good estimation process. We can think of relevant estimates as simple statistical estimators that average the length of all previous relevant tasks. To create such an estimator, we first need to collect a dataset. If you follow a waterfall process, then to get one new data point in your estimation dataset, you need to complete a whole project from start to end. You may need to fail estimates in many projects before you get good at estimating one particular type of project.

Differentiating between Business and Implementation Estimates:

We can look at estimates from two perspectives.

The first one will be familiar to project managers and team leaders who are mostly concerned with project delivery: the implementation perspective. The main goal of estimates in this example is to provide correct expectations regarding how much time and money will be required to build the solution.

Another perspective is closely related to the business goals of the project and is often unseen by the implementation team. Every project is generally backed up by a business model that fixes expectations on revenue increases, customer satisfaction, reduced costs, and so on.

This business model should always be considered when you're creating implementation estimates. In data science projects, business estimations can be included in the project by deriving budget constraints from a business model and creating a set of business metrics that will evaluate the project's performance.

Learning to make time and cost estimates

Using relative estimates is an effective strategy, but it becomes useless if someone asks you, *when exactly will you be able to finish this? Scrum and Kanban do not give you project* estimation tools. In fact, both methodologies argue that making such estimates is

unnecessary. This line of thought is true if your goal is to efficiently complete a project with known deadlines and known budget constraints. However, there are situations where you may need to set budgeting and time constraints yourself.

Let's take a consulting environment as an example. We need to build a custom analytics system for a client. The main task is to estimate the probability of buying a certain product based on the user's profile. This customer needs an entirely new solution that will meet the requirements of various stakeholders from several departments. They also ask you to integrate the solution with various IT systems. They have invited several companies to compete for the project.

The first thing they ask each candidate company is:

How much will this cost and how fast will you be able to build it?

How can we approach this if we know the limitations of absolute estimations?

Let's start with the outline. The outline is a hierarchical list of high-level tasks that you will need to complete. The simplest outline for a waterfall project may look like this:

1. Collect the requirements
2. Implement the requirements
3. Test the system
4. Deploy the system

Using a waterfall project is risky, so we will split the system into several stages, with each going through several successive steps. Depending on the complexity of the stage, you will need to make one or several iterations of the same stage to complete it. In theory, you could try to create an

outline for every two-week sprint, but this is unrealistic because of the ever changing nature of data science projects.

For example, let's look at the outline for requirements collection:

1. Collect the requirements:
    - Software architecture:
        - Nonfunctional requirements specification
        - Nonfunctional requirements implementation strategy
        - Component diagram
        - Integration diagram
    - Functional requirements specification:
    - UI:
        - UI requirements
        - UI mockups
    - Backend services
    - Data analysis and modeling:
        - EDA
        - Creating an experimentation backlog

You will have difficulty decomposing some tasks. Being unsure about an approach to a task is a red flag signaling that you need to communicate with your customers and figure this out. If you do not how many systems you will need to integrate with, you will have a hard time decomposing all the stages and tasks related to integration. In this case, we need to call the customer and quickly discover the necessary information. However, you may have hundreds of such questions during the estimation process, and tracking new information will quickly become ineffective. It is a good idea to prepare a numbered list of questions first. Answers can change during the estimation process, so each question should be assigned a date. Ideally, those questions should be shared in a format that makes

collaboration easy.

When your outline is detailed enough, it is time to design a software architecture proposal.

This is a crucial step because matching outlines with customer requirements is not always

economically viable or even possible from a technological standpoint. You should have at

least a rough idea of what technologies you will use, how they will integrate with the rest of the customer's system, and how your solution should be deployed. If there are any crucial nonfunctional requirements, such as 24/7 availability, the software architect should also think about how to implement them in terms of technology and system design.
Drafting a high-level architecture vision will help explain this outline. Do not hesitate to change the outline if you think that's necessary. Software design is a complex task an experienced engineer should do, so if you do not have deep expertise in designing software solutions, ask for help from someone on your team, or even better, make software design a collaborative effort.

After you've completed the outline and have a software architecture vision, you can start estimating the project. Recommending using simple statistical estimation procedures such as the program evaluation and review technique (PERT) might be a good choice.

In PERT, you give each task a three-point estimate:

- **Optimistic estimate: The time you plan to spend on the task if** everything goes well; minor technical problems and requirements issues may arise.
- **Most likely estimate:** The most realistic estimate you can give for the task.
- **Pessimistic estimate:** The time that's required to finish the task if problems arise.

This includes additional risks for dealing with experiments that have gone wrong, complex debugging sessions, and having long debates with the customer.

Then, you can calculate a simple weighted average to get the final estimate

PERT Estimate = (Optimistic estimate +4 X Most likely estimate + pessimistic estimate) / 6

Calculating the standard deviation is also useful when it comes to making confidence intervals:

$$PERT\ Standard\ Deviation = \frac{Pessimistic\ estimate - Optimistic\ estimate}{6}$$

$Pert\ Estimate \pm 3 \times PERT\ Standard\ Deviation$ will give you a 99.7% confidence interval, meaning that the task will end up somewhere between these numbers with a 99.7% probability. If this range is too wide for you, you can use $2 \times PERT\ Standard\ Deviation$, which will give you a 95.5% confidence interval.

Use data from any finished projects as a base for relative estimation. The more external resources you use for estimation, the more accurate and risk-averse your estimates will become.

Since we are ineluctably bad at estimation, all of the estimates are only a rough idea of your current view of the project implementation plan. Project outlines and estimates should constantly change and be adapted to the current situation. You should periodically check whether the original plan should be changed and updated. If so, convey this to the customer and work through the necessity of the changes. It may be that the customer added several new features to your backlog, thinking that they were present in the original scope.

If this is not the case, negotiate a scope expansion, followed by a budget and deadline extension. If these features are not critical enough, advise the customer to remove them from the backlog. With each completed task on a particular type of project, your experience will grow. As a result, you will be able to anticipate more of the customer's needs and include them in the base plan, which will make estimates more accurate. Store all of the versions of the project estimations so that you can track all scope changes effortlessly.

Project architecture vision should also be robust in terms of changes. The more customized your solution is, the less likely it is that it will create an ideal architecture vision that will survive all scope changes. Plan ahead and include several variation points in the parts of your solution that are the most likely to change. A vacation point is a software component (or a set of software components) that was going to change from the start. A plugin architecture and microservices with fixed contracts are examples of variation points that allow for easy extension or substitution.

Discovering the goals of the estimation process

It is important to keep the end goal in mind while making estimations. You can build a data science system without making a grand plan. Creating estimates and keeping them up to date requires a lot of effort and time. Data science projects are complex and unpredictable, so the more you and your customers believe in your estimates, the more likely they're going to fail. Estimates become more uncertain if your team has no prior experience in building solutions for a new business domain or if you are trying to apply new types of algorithms or use new technologies.

Having a fine-grained view of how to achieve the end goal is useful. In contrast, relying on the exact calculations of how long it will take you, or using extremely detailed outlines, is not. Use estimates wisely; they will help you align your implementation plans with customer demand.