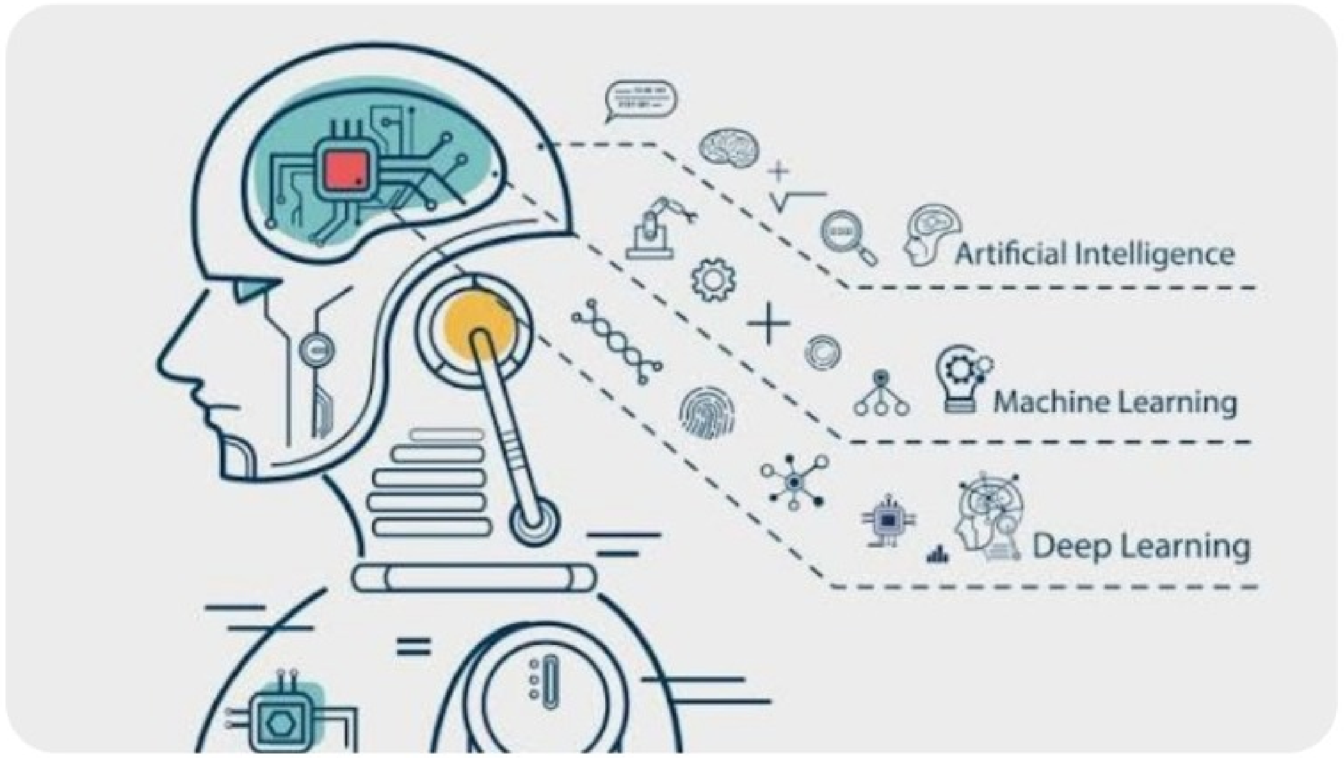
EBOOK

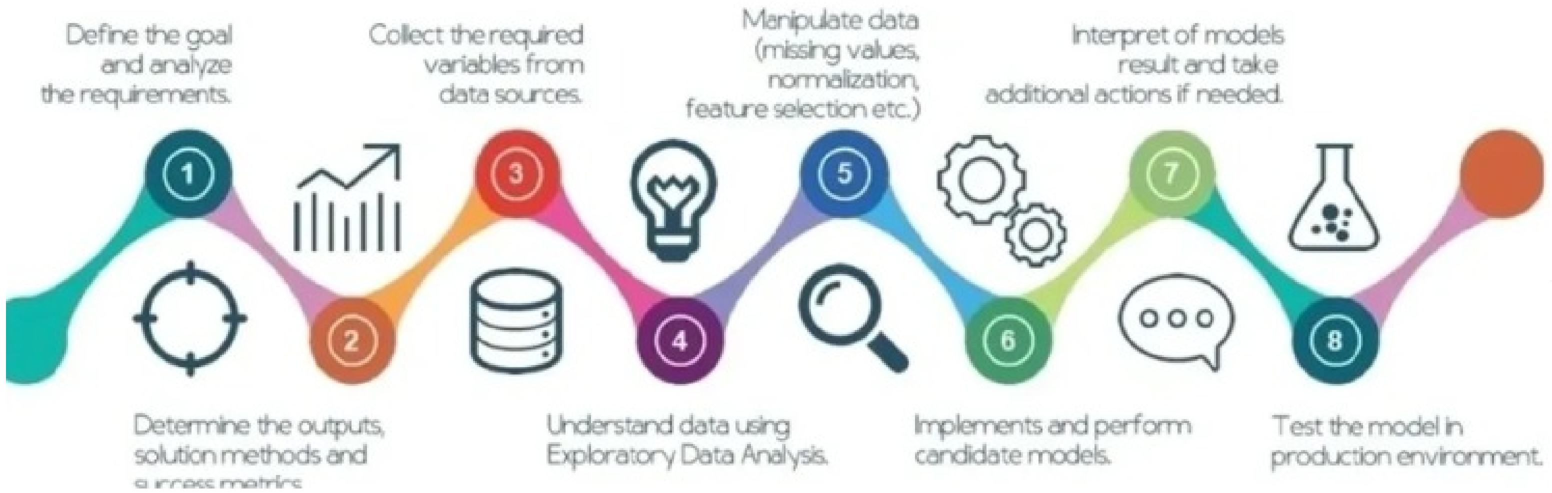
METHODOLGY OF KEEPGOING.AI

Machine Learning Engineering for the Real World

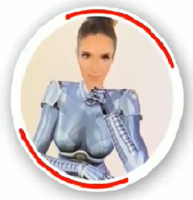
A step-by-step guide to take your ML projects from planning to production



SYSTEM DEVELOPMENT SERVICES



Define the goal and analyze the requirements.
Collect the required variables from data sources.
Manipulate data (missing values, normalization, feature selection etc.)
Interpret of models result and take additional actions if needed.
1
2
3
4
5
6
7
8
Determine the outputs, solution methods and success metrics
Understand data using Exploratory Data Analysis.
Implements and perform candidate models.
Test the model in production environment.



keepgoing.ai

Introduction

In this eBook, you will learn about the key components of machine learning engineering, how to successfully implement large-scale machine learning projects from scoping to deployment, and the key teams and personas involved in executing successful machine learning initiatives at companies.

keepgoing.ai

Contents

keepgoing.ai

SECTION

01

ML engineering as a concept

keepgoing.ai

SECTION 1

ML engineering as a concept

Machine learning (ML) is exciting. To the layperson, it brings with it the promise of seemingly magical abilities of soothsaying, uncovering mysterious and miraculous answers to difficult problems. ML makes money for companies, it autonomously tackles overwhelmingly large tasks, and it removes the burdensome task of monotonous work. To state the obvious, though, it's challenging. Using thousands of algorithms and requiring a diverse skill set ranging from data engineering (DE) to advanced statistical analysis and visualization, the work of a professional ML practitioner is complex and truly intimidating.

ML engineering is the concept of applying a system around this staggering level of complexity. It is a set of standards, tools, processes and methodology that aims to minimize time wasted on abandoned, misguided or irrelevant work when solving a business problem or need. It, in essence, is the roadmap to creating ML-based systems that can not only be deployed to production, but also maintained and updated for years into the future, allowing businesses to reap the rewards in efficiency, profitability and accuracy that ML in general has proven to provide (when done correctly).

This eBook is a roadmap to guide you through this system. As shown in Figure 1.1, it entails a proven set of processes about the planning phase of project work — including navigating the difficult and confusing translation of business needs into the language of ML work. From that, it covers a standard methodology of experimentation work, focusing on the tools and coding standards for creating a minimum viable product (MVP) that will be comprehensive and maintainable. Finally, it covers the various tools, techniques and nuances involved in crafting production-grade maintainable code that is both extensible and easy to troubleshoot.

MACHINE LEARNING ENGINEERING

The most critical — and underrated — element of the machine learning lifecycle

keepgoing.ai

However, ML engineering is not exclusively about the path shown in Figure 1.1. It is also about the methodology within each of these stages, which can make or break a project. It is the way in which a data science team talks to a business about a problem, the manner in which research is done, the details of experimentation, the way the code is written, and the multitude of tools and technology that are employed along the roadmapped path that can greatly reduce the worst fate of any project: abandonment.

The end goal of ML work is, after all, about solving a problem. By embracing the concepts of ML engineering and following the road of effective project work, the end goal of getting a useful modeling solution can be shorter and far cheaper and have a much higher probability of succeeding than if you just “wing it” and hope for the best.

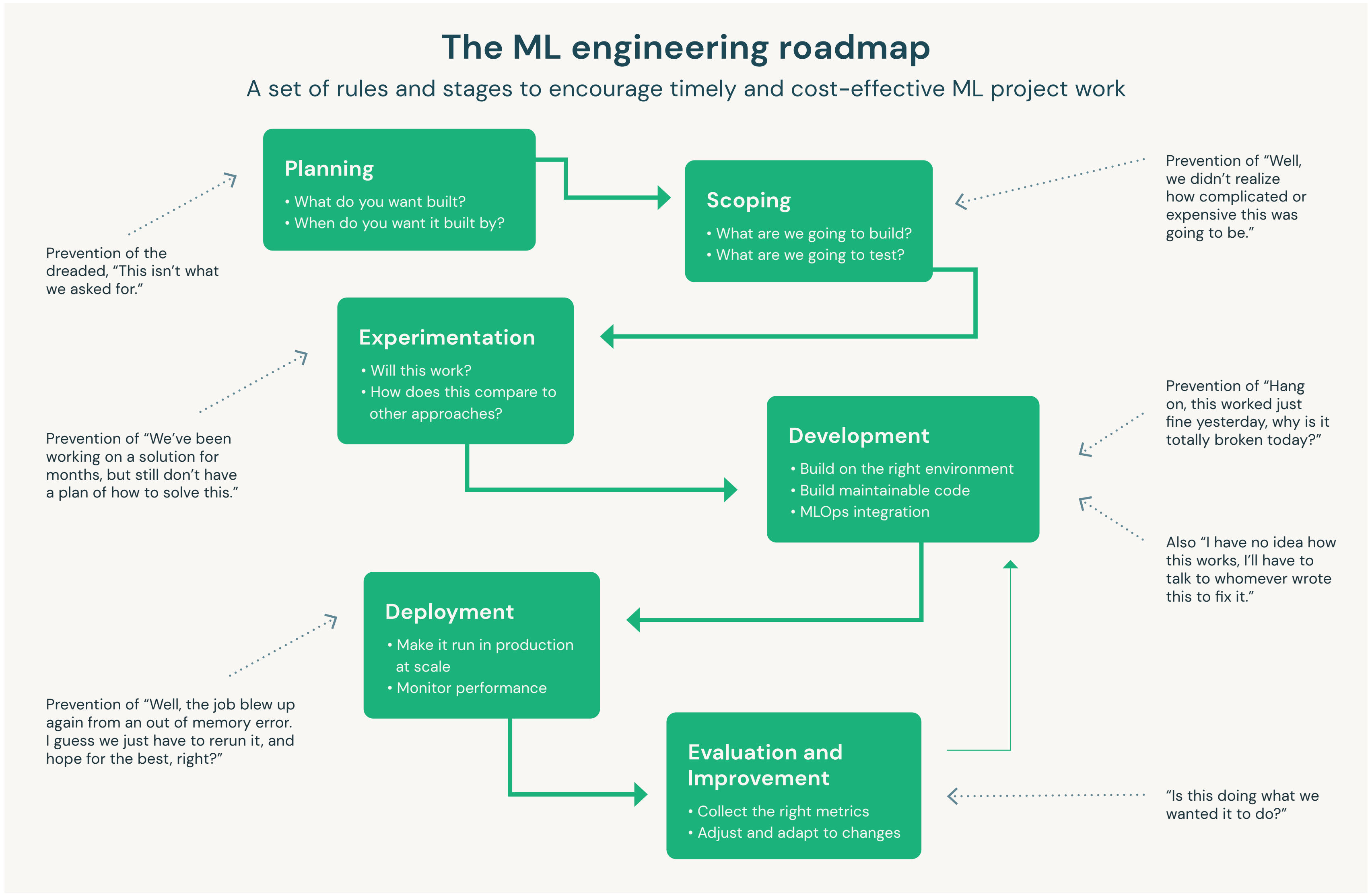

The ML engineering roadmap
A set of rules and stages to encourage timely and cost-effective ML project work
Planning
• What do you want built?
• When do you want it built by?
Prevention of the dreaded, “This isn’t what we asked for.”
Scoping
• What are we going to build?
• What are we going to test?
Prevention of “Well, we didn’t realize how complicated or expensive this was going to be.”
Experimentation
• Will this work?
• How does this compare to other approaches?
Prevention of “We’ve been working on a solution for months, but still don’t have a plan of how to solve this.”
Development
• Build on the right environment
• Build maintainable code
• MLOps integration
Prevention of “Hang on, this worked just fine yesterday, why is it totally broken today?”
Also “I have no idea how this works, I’ll have to talk to whomever wrote this to fix it.”
Deployment
• Make it run in production at scale
• Monitor performance
Prevention of “Well, the job blew up again from an out of memory error. I guess we just have to rerun it, and hope for the best, right?”
Evaluation and Improvement
• Collect the right metrics
• Adjust and adapt to changes
“Is this doing what we wanted it to do?”

Figure 1.1
The ML engineering roadmap shows the proven stages of work involved in creating successful ML solutions. While some projects may require additional steps (particularly if working with additional engineering teams), these are the fundamental steps that should be involved in any ML-based project.

keepgoing.ai

SECTION 1: ML ENGINEERING AS A CONCEPT

1.1 Why ML engineering?

To put it most simply, ML is *hard*. It's even harder to do correctly in the sense of serving relevant predictions, at scale, with reliable frequency. With so many specialties existing in the field (NLP, forecasting, deep learning, traditional linear and tree-based modeling, etc.), an enormous focus on active research and so many algorithms that have been built to solve specific problems, it's remarkably challenging to learn more than a tiny fraction of all there is to learn. Coupling that complexity with the fact that one can develop a model on everything from a Raspberry Pi to an enormous NVIDIA GPU cluster, the very platform complexities that are out there present an entirely new set of information that no one person could have enough time in their life to learn.

There are also additional realms of competency that a data scientist is expected to be familiar with. They include midlevel data engineering skills (you have to get your data for data science somewhere, right?) as well as skills in software development, project management, visualization and presentation — and the list continues to grow, making it rather daunting to gain all the necessary experience. So it shouldn't be surprising, considering all of this, why attaining all the required skills to create production-grade ML solutions is beyond the reach of most individuals.

The aim of ML engineering is not to iterate through the lists of such skills and require that a data scientist master each of them. Instead, it's to treat it as a collection of certain aspects of those skills, carefully crafted to be relevant to data scientists, all with the goal of increasing the chances of *getting an ML project into production* and to make sure that it's not a solution that needs constant maintenance and intervention to keep running.

An ML engineer, after all, doesn't need to be able to create applications and software frameworks for generic algorithmic use cases. They're also not likely to be writing their own large-scale streaming ingestion ETL pipelines. Nor do they need to be able to create detailed and animated front-end visualizations in JavaScript.

An ML engineer needs to know *just enough software development skills* to be able to write modular code and to implement unit tests. They don't need to know about the intricacies of nonblocking asynchronous messaging brokering. They need *just enough data engineering skills* to build (and schedule the ETL for) feature data sets for their models, but they don't need to construct a PB-scale streaming ingestion framework. They need *just enough visualization skills* to create plots and charts that communicate clearly what their research and models are doing, but they don't have to develop dynamic web apps with complex UX components. They also need *just enough project management experience* to know how to properly define, scope and control a project to solve a problem, but they don't need to go through a PMP certification.

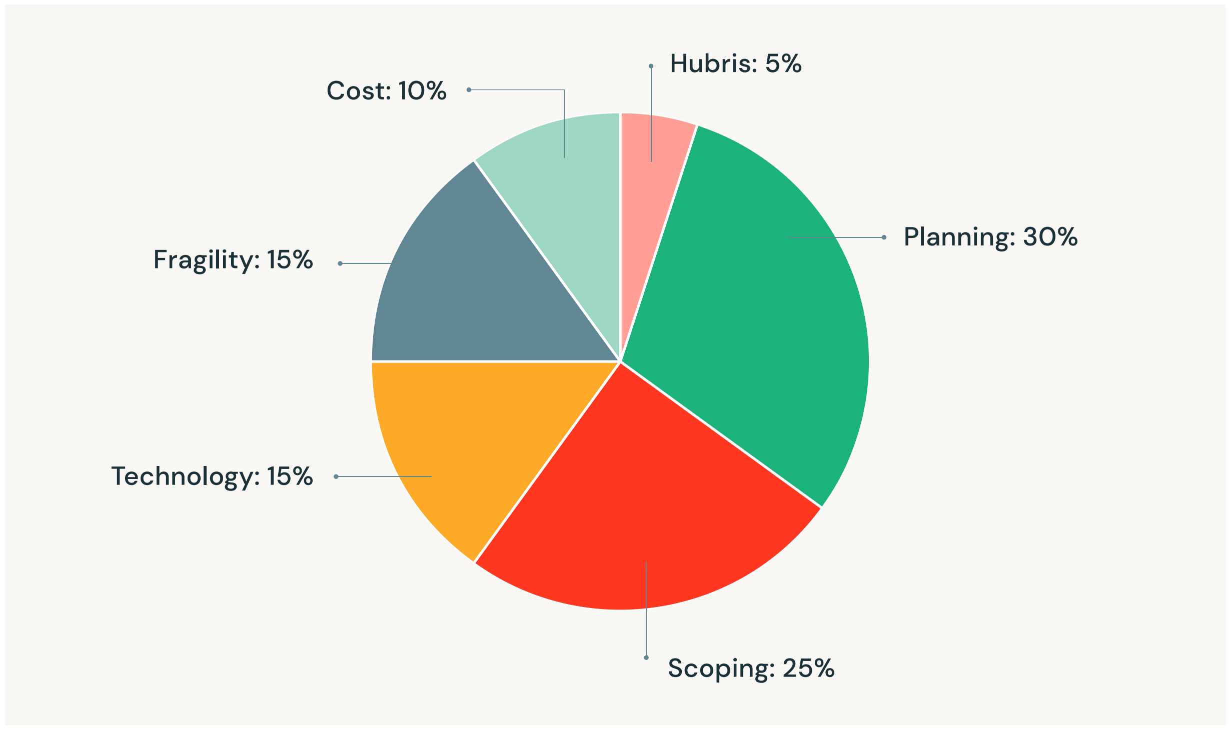


Figure 1.2
Primary reasons for ML project failures. Figure 1.2 shows some rough estimates of the primary reasons why projects fail. Most commonly, data science teams are either inexperienced with using a large-scale production-grade model to solve a particular need or simply fail to understand what the desired outcome from the business is.

Despite many companies going all in on ML, hiring massive teams of highly compensated data scientists and devoting huge amounts of financial and temporal resources, these projects end up failing at incredibly high rates. This eBook covers the six major causes of project failure and why they result in so many projects failing, being abandoned or taking longer than necessary to reach production. In each section, we will show the solutions to these common problems and explain the processes involved in reaching those solutions so you can significantly lower the chances of your project getting derailed.

These issues are not the result of malicious intent. Rather, they are due in large part to the fact that most ML projects are incredibly challenging and complex, and are composed of algorithmic software tooling that is hard to explain to a layperson (hence the breakdowns in communication with business units that most projects endure). With such complex solutions in play, so many moving parts and a world of corporations trying to win in this new data-focused arms race and profit from ML as quickly as possible, it's no wonder that the perilous journey of taking a solution to a point of stability in production fails so frequently.

This book is meant to show how these elements pose a risk to projects and to teach the tools that help minimize the risk of each. By focusing on each of these areas in a conscientious and deliberate manner, many of these risks can be significantly mitigated, if not eliminated entirely.

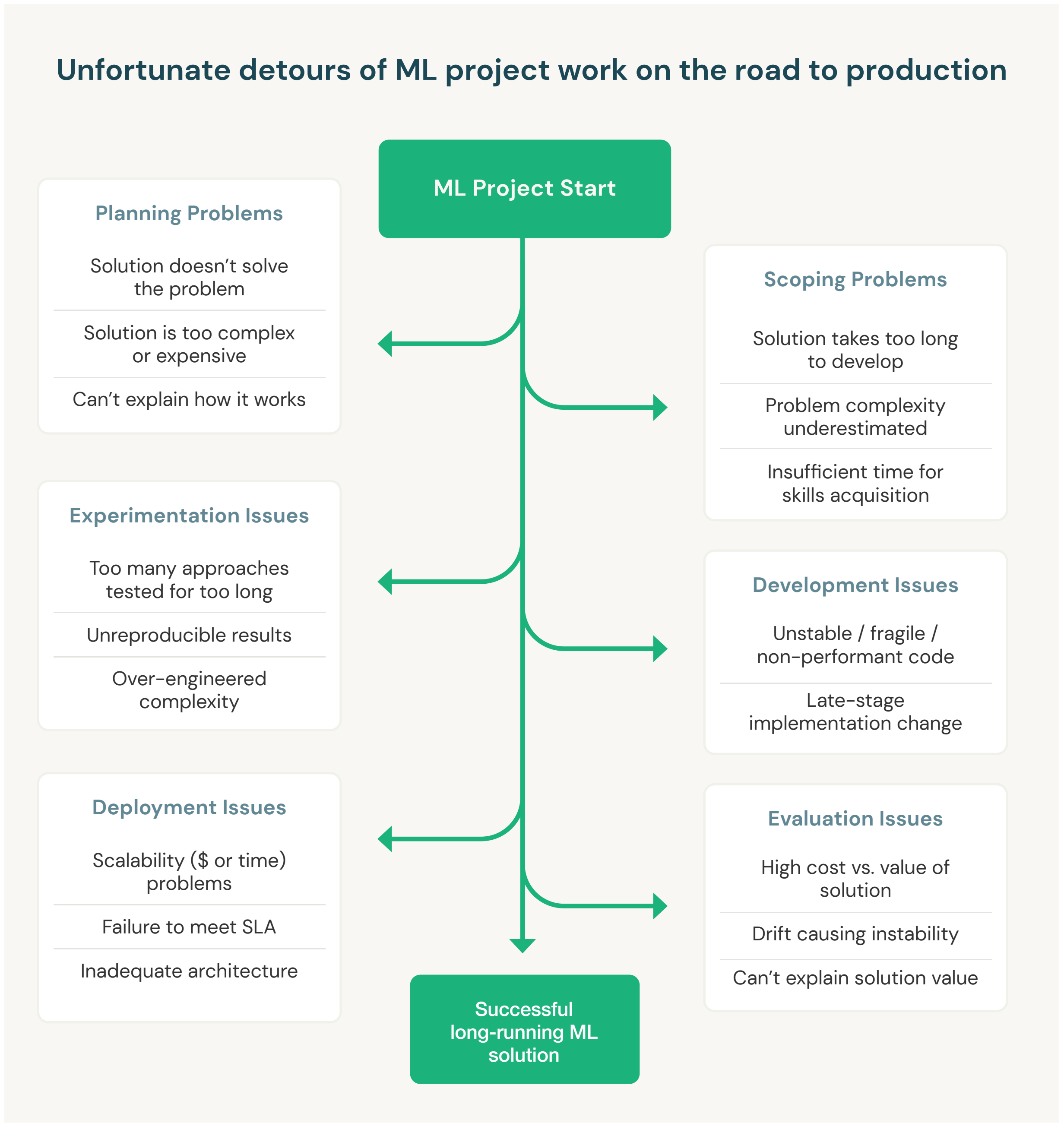

Unfortunate detours of ML project work on the road to production
ML Project Start
Planning Problems
Solution doesn't solve the problem
Solution is too complex or expensive
Can't explain how it works
Scoping Problems
Solution takes too long to develop
Problem complexity underestimated
Insufficient time for skills acquisition
Experimentation Issues
Too many approaches tested for too long
Unreproducible results
Over-engineered complexity
Development Issues
Unstable / fragile / non-performant code
Late-stage implementation change
Deployment Issues
Scalability ($ or time) problems
Failure to meet SLA
Inadequate architecture
Evaluation Issues
High cost vs. value of solution
Drift causing instability
Can't explain solution value
Successful long-running ML solution

Figure 1.3
The branching paths of failure in the vast majority of ML projects. Nearly all ML solutions that don't focus on these six core areas have a much higher chance of being abandoned either before production or shortly after running in production.

In Figure 1.3, you'll see a representation of the path that all of us are moving on when we employ ML to solve a problem. From the outset of a project to its planned successful, long running and maintainable state, the journey is fraught with detours that can spell the termination of our hard work. However, by focusing on building up our knowledge, skills and utilization of processes and tooling, we can generally avoid these six major problematic areas — or, at the very least, we can address them in a way that won't cause a complete failure of a project.

ML engineering is designed to address each of the primary failure modes shown in Figure 1.3. Eliminating the chances of failure is at the heart of this methodology. This is done through processes that lead to better decisions, ease communication with internal customers, eliminate rework during the experimentation and development phases, create code bases that can be easily maintained, and bring a best-practices approach to any project work that is heavily influenced by data science work. Just as software engineers decades ago refined their processes from large-scale waterfall implementations to a more flexible and productive agile process, ML engineering seeks to define a new set of practices and tools that will optimize the wholly unique realm of software development for data scientists.

SECTION 1: ML ENGINEERING AS A CONCEPT

1.2 The core tenets of ML engineering

Now that you have a general idea of what defines ML engineering, we can focus on the key elements:

Planning

Neglecting to plan out projects thoroughly is the biggest cause of their failure by far — and it's one of the most demoralizing ways for them to be canceled. Imagine for a moment that you're the first data scientist hired by your company. In your first week, an executive from marketing approaches you, explaining (in their terms) a serious business issue that they are facing. They need to figure out an efficient means of communicating to customers through email about upcoming sales they might be interested in. With very little additional detail provided to you, the executive merely says, "I want to see the click and open rates go up."

If this were the only information supplied and if members of the marketing team answered your repeated queries by simply stating the same end goal of increasing the clicking and opening rates, there would seem to be a limitless number of avenues to pursue. Left to your own devices, do you:

- Focus on content recommendation and craft custom emails for each user?
- Provide predictions with an NLP-backed system that will craft relevant subject lines for each user?
- Attempt to predict a list of products most relevant to the customer base to put on sale each day?

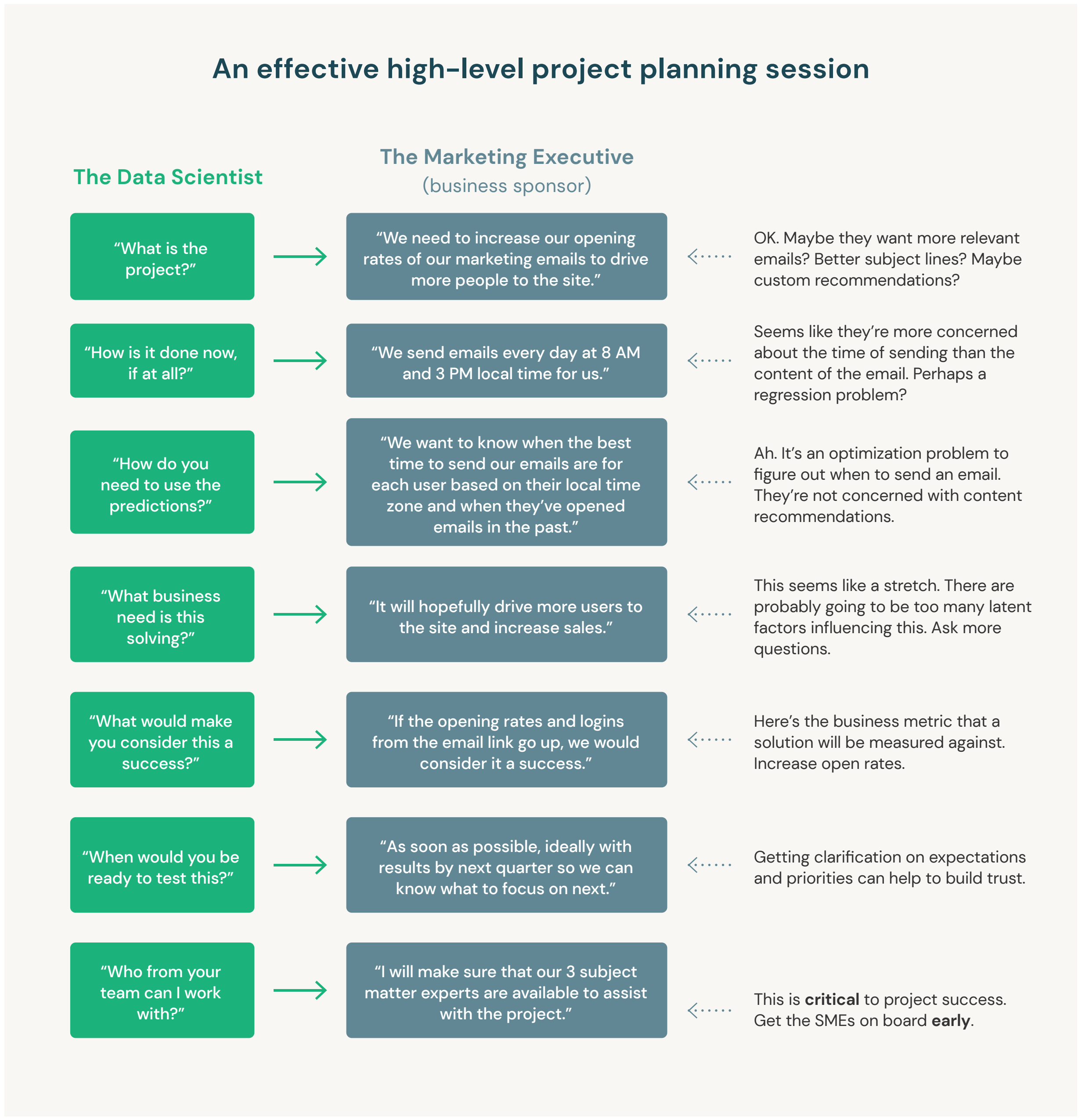
With so many options of varying complexity and approaches, and with very little guidance, the possibility of creating a solution that is aligned with the expectations of the executive is highly unlikely.

If a proper planning discussion had taken place, the true expectation might be revealed: a prediction for each user about when they would be most inclined to read emails. The executive simply wants to know when someone is most likely to not be at work, commuting or sleeping so that they can send batches of emails throughout the day to different cohorts of customers.

The sad reality is that many ML projects start off this way. There is frequently very little communication with regard to project initiation, and the general expectation is that "the data science team will figure it out." However, without the proper guidance on what needs to be built, how it needs to function and what the end goal of the predictions is, the project is almost doomed to failure.

After all, what would have happened if an entire content recommendation system had been built for that use case, with months of development and effort put in, when a very simple analytics query based on IP geolocation was what was really needed? The project would not only be canceled, but there would likely be many questions from on high as to why this system was built and why development was so expensive.

If we were to look at a very simplified planning discussion at an initial phase, as shown in Figure 1.4, we can see how just a few careful questions and clear answers can give the one thing that every data scientist should be looking for in this situation: a quick win.



An effective high-level project planning session

The Data Scientist	The Marketing Executive (business sponsor)	
"What is the project?"	"We need to increase our opening rates of our marketing emails to drive more people to the site."	OK. Maybe they want more relevant emails? Better subject lines? Maybe custom recommendations?
"How is it done now, if at all?"	"We send emails every day at 8 AM and 3 PM local time for us."	Seems like they're more concerned about the time of sending than the content of the email. Perhaps a regression problem?
"How do you need to use the predictions?"	"We want to know when the best time to send our emails are for each user based on their local time zone and when they've opened emails in the past."	Ah. It's an optimization problem to figure out when to send an email. They're not concerned with content recommendations.
"What business need is this solving?"	"It will hopefully drive more users to the site and increase sales."	This seems like a stretch. There are probably going to be too many latent factors influencing this. Ask more questions.
"What would make you consider this a success?"	"If the opening rates and logins from the email link go up, we would consider it a success."	Here's the business metric that a solution will be measured against. Increase open rates.
"When would you be ready to test this?"	"As soon as possible, ideally with results by next quarter so we can know what to focus on next."	Getting clarification on expectations and priorities can help to build trust.
"Who from your team can I work with?"	"I will make sure that our 3 subject matter experts are available to assist with the project."	This is **critical** to project success. Get the SMEs on board **early**.

Figure 1.4
A simplified planning discussion to get to the root of what an internal customer — in this case, the marketing executive who wants high open rates on their emails — actually needs for a solution

As Figure 1.4 shows, the problem at hand is not at all in the list of original assumptions that were made. There is no talk about the content of the emails, relevancy to the subject line or the items in the email. It's a simple analytical query to figure out which time zone customers are in and to analyze historic openings in local times for each customer. By taking a few minutes to plan and understand the use case fully, weeks (if not months) of wasted effort, time and money can be saved.

By focusing on what will be built and why it needs to be built, both the data science team and the business are able to guide the discussion more fruitfully. Eschewing a conversation focused on how it will be built keeps the data science members of the group focused on the problem. Ignoring when it will be built helps the business keep their focus aligned on the needs of the project.

Avoiding any discussion of implementation details at this stage allows the data science team to focus on the problem, which is critical. Keeping the esoteric details of algorithms and solution design out of discussions with the larger team allows the business unit members to stay engaged. After all, they really don't care how many eggs go into the mix, what color the eggs are or even what species laid the eggs — they just want to eat the cake when it's done.

Scoping and research

The focus of scoping and research needs to be on the two biggest questions that internal customers (the business) have about the project.

- Is this going to solve my problem?
- How long is this going to take?

Let's look at another potentially familiar scenario to discuss polar opposite ways that this stage of ML project development can go awry. For this example, there are two separate data science teams at a company — Team A in Figure 1.5 and Team B in Figure 1.6 — each being pitted against one another to develop a solution to an escalating incidence of fraud occurring with the company's billing system.

Team A's research and scoping process is illustrated in Figure 1.5.

Team A is composed of mostly junior data scientists, all of whom entered the workforce without an extensive period in academia. They proceed, upon learning the details of the project and what is expected of them, to immediately go to blog posts. The team searches the internet for "detecting payment fraud" and "fraud algorithms," finding hundreds of results from consultancy companies, a few extremely high-level blog posts from similar junior data scientists who have likely never put a model into production, and some open source — and very rudimentary — data examples.

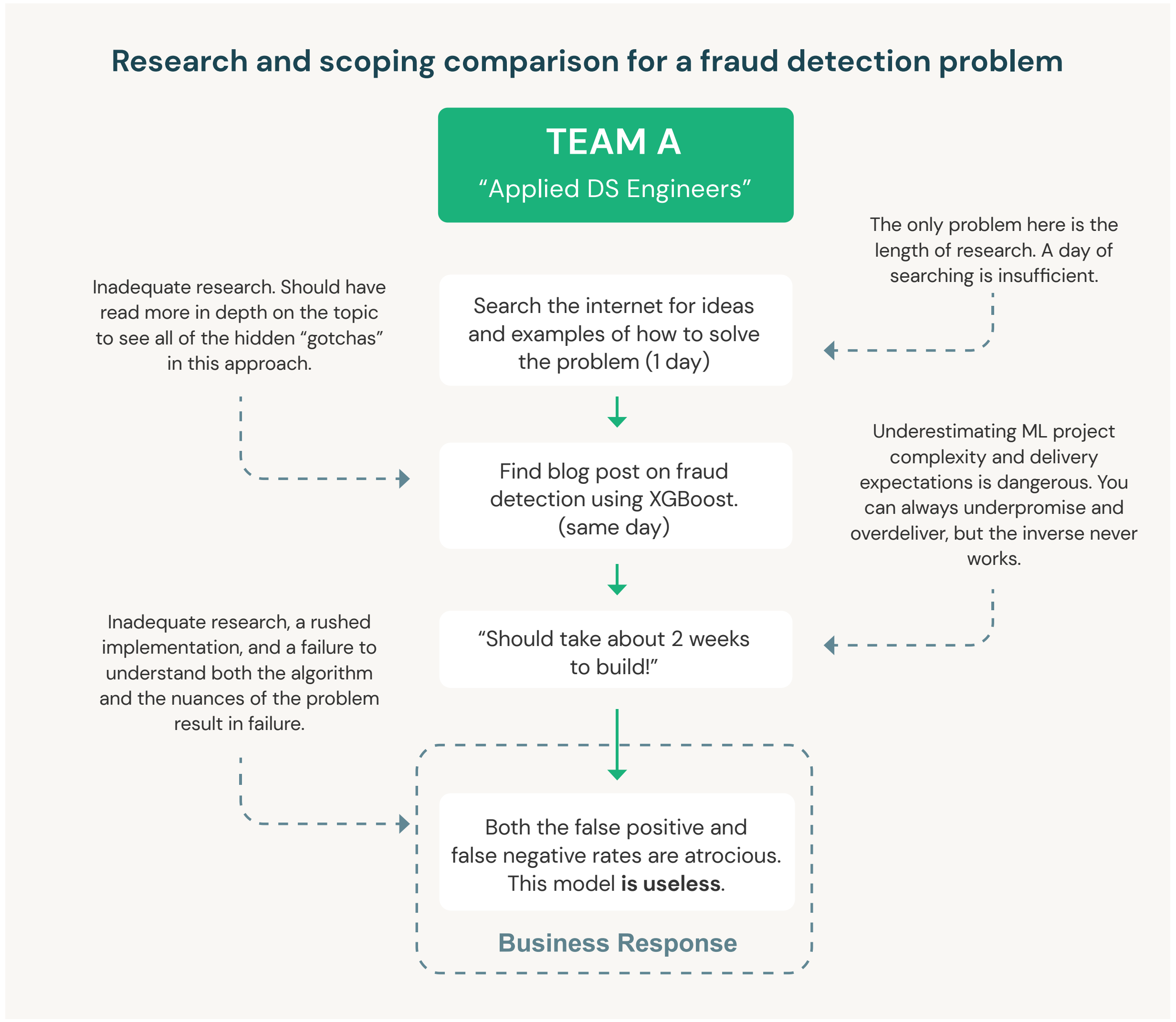

Research and scoping comparison for a fraud detection problem
TEAM A
"Applied DS Engineers"
Search the internet for ideas and examples of how to solve the problem (1 day)
Find blog post on fraud detection using XGBoost. (same day)
"Should take about 2 weeks to build!"
Both the false positive and false negative rates are atrocious. This model **is useless**.
Business Response
Inadequate research. Should have read more in depth on the topic to see all of the hidden "gotchas" in this approach.
Inadequate research, a rushed implementation, and a failure to understand both the algorithm and the nuances of the problem result in failure.
The only problem here is the length of research. A day of searching is insufficient.
Underestimating ML project complexity and delivery expectations is dangerous. You can always underpromise and overdeliver, but the inverse never works.

Figure 1.5
Research and scoping of a fraud detection problem for a junior team of well-intentioned but inexperienced data scientists

Team B's research and scoping, shown in Figure 1.6, stands in contrast.

Team B is filled with a group of Ph.D. academic researchers. With their studious approach to research and the vetting of ideas, their first actions are to dig into published papers on the topic of fraud modeling. Spending several days reading through journals and papers, they are now armed with a large collection of theory encompassing some of the most cutting-edge research being done on detecting fraudulent activity.

If you were to ask either of these teams what the level of effort is to produce a solution, you would get wildly divergent answers. Team A would likely state that it would take about two weeks to build their XGBoost binary classification model (they mistakenly believe that they already have the code, after all, from the blog post that they found).

Team B would tell a vastly different tale. They'd estimate it would take several months to implement, train and evaluate the novel deep learning structure that they found in a highly regarded whitepaper whose proven accuracy for the research was significantly better than any perforce implemented algorithm for this use case.

But with their approaches to scoping and research, these two teams — polar opposites — would both see their projects fail, although for two completely different reasons. Team A would have a project failure because the solution to the problem is significantly more complex than the example shown in the blog post they found (the class imbalance issue alone is too challenging of a topic to effectively document in the short space of a blog post). Team B, even though their solution would likely be extremely accurate, would never be allocated resources to build such a risky solution as an initial fraud detection service at the company (although it would be a great candidate for a version 2.0 implementation).

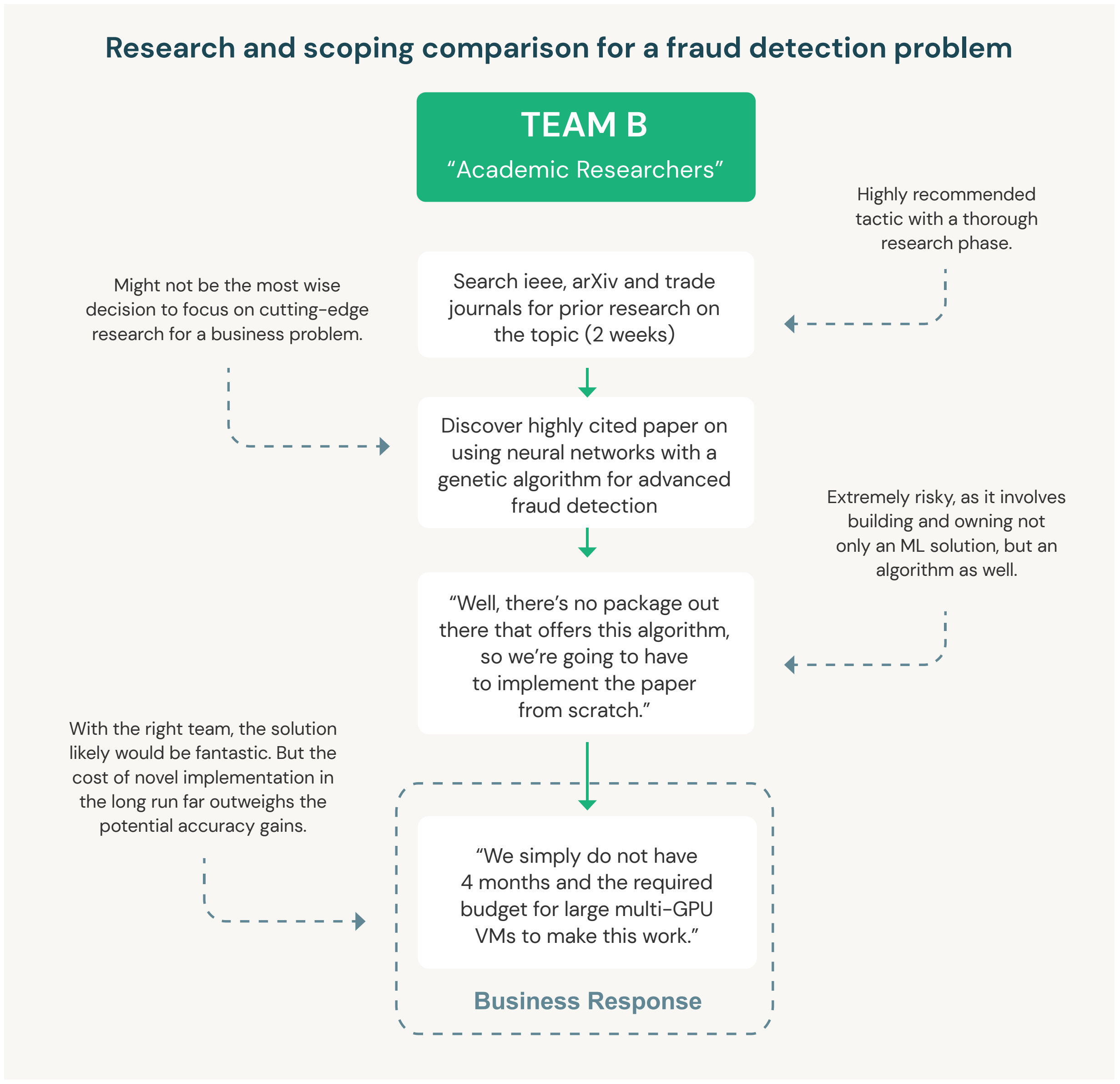

Research and scoping comparison for a fraud detection problem
TEAM B
"Academic Researchers"
Highly recommended tactic with a thorough research phase.
Might not be the most wise decision to focus on cutting-edge research for a business problem.
Search ieee, arXiv and trade journals for prior research on the topic (2 weeks)
Discover highly cited paper on using neural networks with a genetic algorithm for advanced fraud detection
Extremely risky, as it involves building and owning not only an ML solution, but an algorithm as well.
"Well, there's no package out there that offers this algorithm, so we're going to have to implement the paper from scratch."
With the right team, the solution likely would be fantastic. But the cost of novel implementation in the long run far outweighs the potential accuracy gains.
"We simply do not have 4 months and the required budget for large multi-GPU VMs to make this work."
Business Response

Figure 1.6
Research and scoping for an academia-focused group of researchers for the fraud detection problem

Project scoping for ML is incredibly challenging. Even for the most seasoned of ML veterans, making a conjecture about how long a project will take, which approach is going to be most successful and the amount of resources that will need to be involved is a futile and frustrating exercise. The risk associated with making erroneous claims is fairly high, but there are means of structuring proper scoping and solution research that can help minimize the chances of an estimation being wildly off.

Most companies have a mix of the types of people in the hyperbolic scenario previously mentioned. There are academics whose sole goal is to further the advancement of knowledge and research into algorithms, paving the way for future discoveries from within industry. There are also "applications of ML" engineers who just want to use ML as a tool to solve a business problem. It's very important to embrace and balance both aspects of these philosophies toward ML work, strike a compromise during the research and scoping phase of a project, and know that the middle ground here is the best path to trod upon to ensure that a project actually makes it to production.

EXPERIMENTATION

In the experimentation phase, the largest cause of project failure is either experimentation that takes too long (testing too many things or spending too long fine-tuning an approach) or an underdeveloped prototype that is so abysmally bad that the business decides to move on to something else.

An example in section 1.2.2 illustrates how these two approaches might play out at a company that is looking to build an image classifier for detecting products on retail store shelves. The experimentation paths that the two groups take (representing the extreme polar opposites of experimentation) are shown in Figures 1.7 and 1.8.

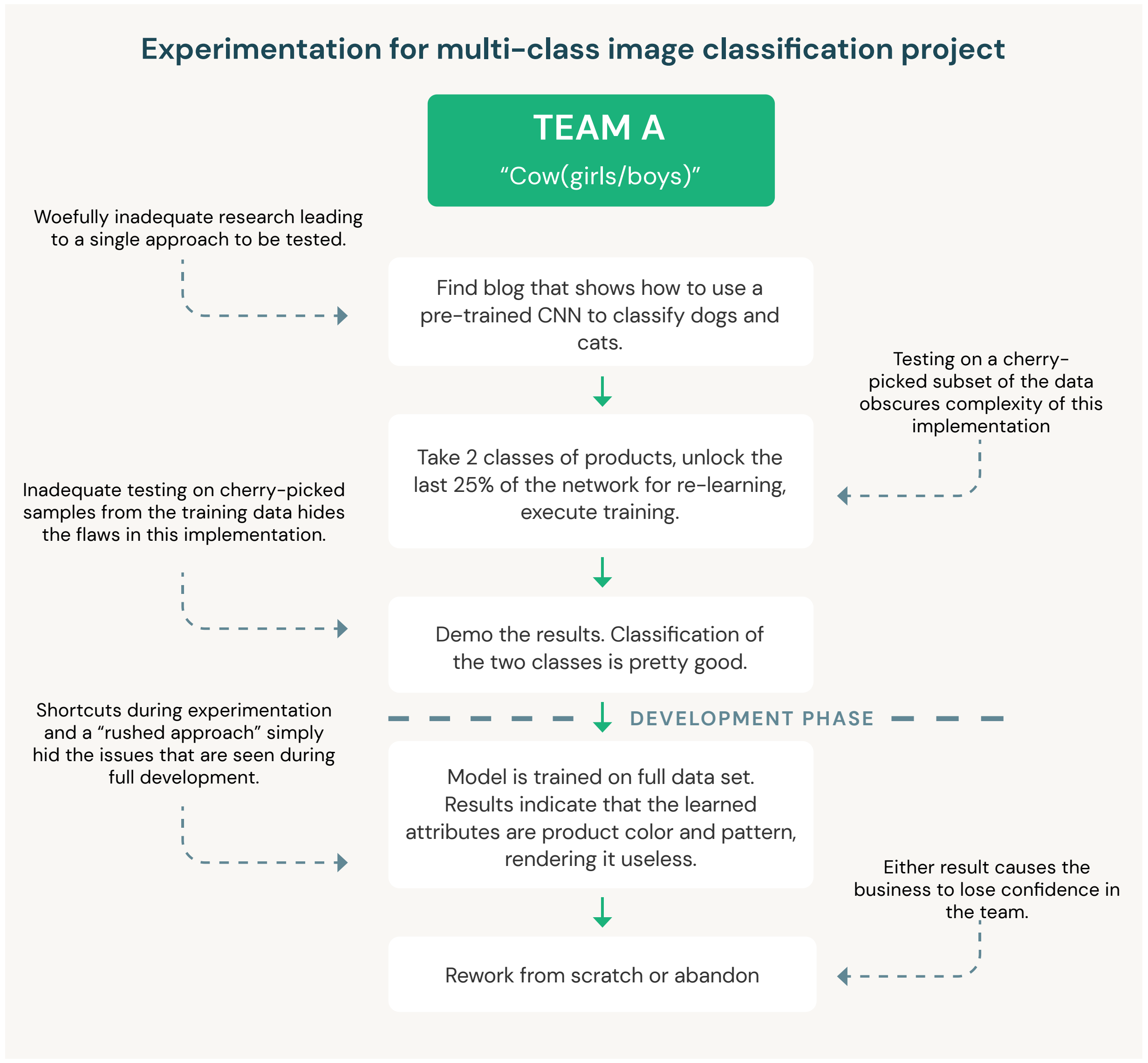

Experimentation for multi-class image classification project
TEAM A
"Cow(girls/boys)"
Woefully inadequate research leading to a single approach to be tested.
Find blog that shows how to use a pre-trained CNN to classify dogs and cats.
Testing on a cherry-picked subset of the data obscures complexity of this implementation
Take 2 classes of products, unlock the last 25% of the network for re-learning, execute training.
Inadequate testing on cherry-picked samples from the training data hides the flaws in this implementation.
Demo the results. Classification of the two classes is pretty good.
Shortcuts during experimentation and a "rushed approach" simply hid the issues that are seen during full development.
DEVELOPMENT PHASE
Model is trained on full data set. Results indicate that the learned attributes are product color and pattern, rendering it useless.
Either result causes the business to lose confidence in the team.
Rework from scratch or abandon

Figure 1.7
A rushed experimentation phase by a team of inexperienced data scientists

Team A in Figure 1.7 is an exceedingly hyperbolic caricature of an exceptionally inexperienced data science team, performing only the most cursory of research. Using the single example blog post that they found regarding image classification tasks, they copy the example code, use the exact pretrained TensorFlow-Keras model cited in the blog, retrain the model on only a few hundred images of just two of the products (out of many thousands), and demonstrate a fairly solid result in classification for the holdout images from these two classes.

But because they didn't do thorough research, they were unable to understand the limitations that were in the model they chose. With their rushed approach to creating a demo to show how well they could classify their own images, they chose a too-simplistic test of only two classes. With cherry-picked results and a woefully inadequate evaluation of the approach, this project would likely fail early in the development process (if someone on their leadership team was checking in on their progress), or late into the final delivery phases before production scheduling (when the business unit's internal customer could see just how badly the approach was performing). Either way, using this rushed and lazy approach to testing will nearly always end in a project that is either abandoned or canceled.

Team B's approach to this problem is shown in Figure 1.8.

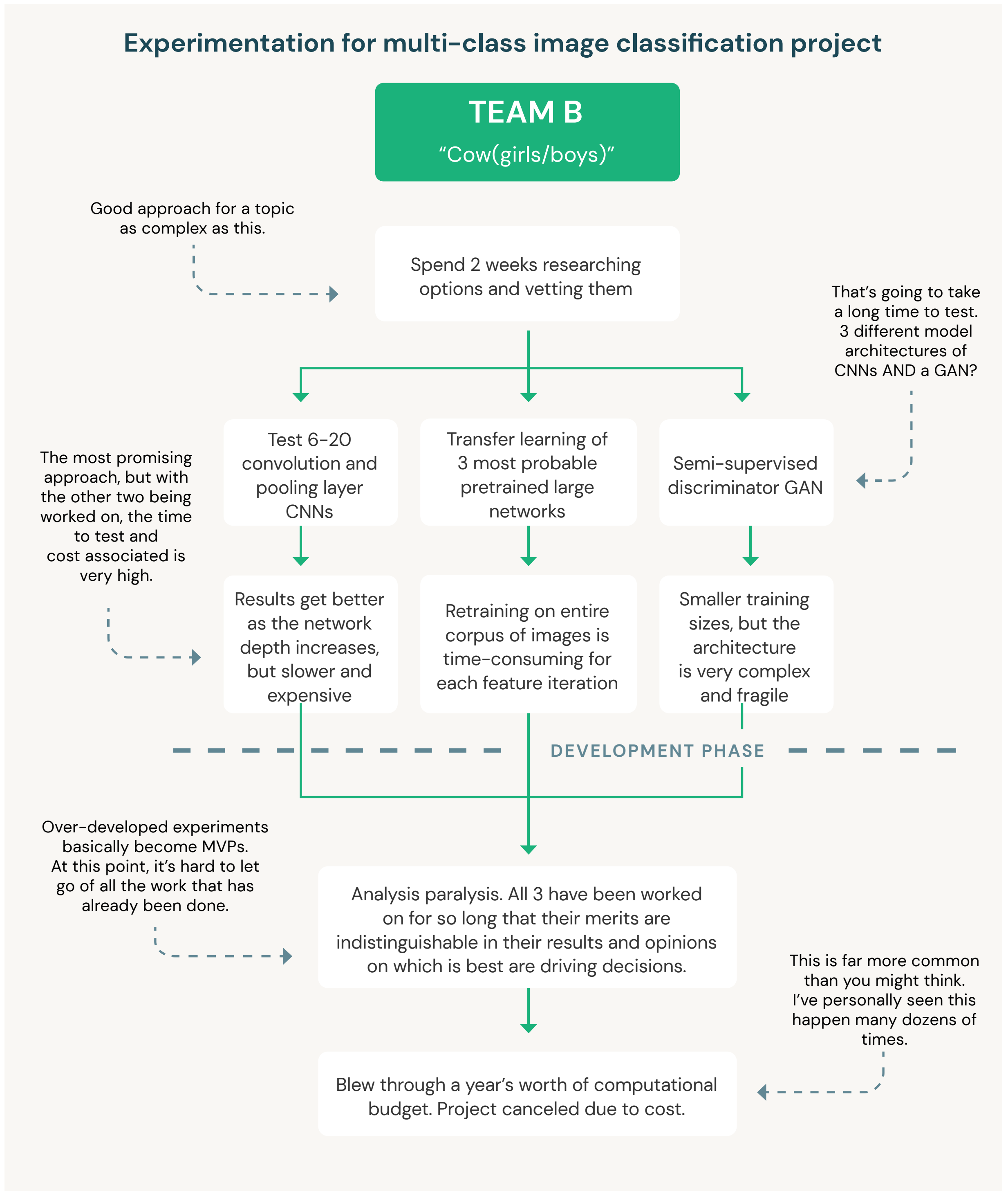

Experimentation for multi-class image classification project
TEAM B
"Cow(girls/boys)"
Good approach for a topic as complex as this.
Spend 2 weeks researching options and vetting them
That's going to take a long time to test. 3 different model architectures of CNNs AND a GAN?
The most promising approach, but with the other two being worked on, the time to test and cost associated is very high.
Test 6-20 convolution and pooling layer CNNs
Transfer learning of 3 most probable pretrained large networks
Semi-supervised discriminator GAN
Results get better as the network depth increases, but slower and expensive
Retraining on entire corpus of images is time-consuming for each feature iteration
Smaller training sizes, but the architecture is very complex and fragile
DEVELOPMENT PHASE
Over-developed experiments basically become MVPs. At this point, it's hard to let go of all the work that has already been done.
Analysis paralysis. All 3 have been worked on for so long that their merits are indistinguishable in their results and opinions on which is best are driving decisions.
This is far more common than you might think. I've personally seen this happen many dozens of times.
Blew through a year's worth of computational budget. Project canceled due to cost.

Figure 1.8
An overly thorough experimentation phase that effectively became the build-out of three separate MVPs for the project

Team B, as illustrated in Figure 1.8, is the polar opposite of team A. They're an example of the "pure researchers" — people who, even though they currently work for a company, still behave as though they are defining their doctoral thesis. Their approach to solving this problem is to spend weeks searching through cutting-edge papers, reading journals and understanding the theory involved in various convolutional neural network (CNN) approaches. They've settled on three broad potential solutions, each consisting of several tests that need to run and be evaluated against the entire collection of their training image data set.

It isn't the depth of research that failed them in this case. The research was appropriate for this use case, after all. The problem was that they were simply trying too many things. Varying the structure and depth of a custom-built CNN requires dozens (if not hundreds) of iterations to "get right" for the use case they're trying to solve. This is work that should be scoped into the development stage of the project, not during evaluation. Instead of doing an abbreviated adjudication of the custom CNN, they decided to test out transfer learning of three large pretrained CNNs, as well as building a generative adversarial network (GAN) to get semi-supervised learning to work on the extremely large corpus of classes that are needed to be classified.

Team B quite simply took on too much work for an experimentation phase. What they're left with at the point that they need to show demonstrations of their approaches is nothing more than decision paralysis and a truly staggering cloud services GPU VM bill. With no real conclusion on the best approach and such a large amount of money already spent on the project, the chances that the entire project will be scrapped is incredibly high.

While not the leading cause of project failure, an experimentation phase can, if done incorrectly, stall or cancel an otherwise great project. The approaches used by our imaginary teams are extreme examples, and while neither one is appropriate, the best course of action is a moderate approach between the two.

DEVELOPMENT

While not precisely a major factor for getting a project canceled directly, having a poor development practice for ML projects can manifest itself in a multitude of ways that can completely kill a project. It's usually not as directly visible as some of the other leading causes, but having a fragile and poorly designed code base and poor development practices can actually make a project harder to work on, easier to break in production and far harder to improve as time goes on.

For an example, let's look at a rather simple and frequent modification situation that comes up during the development of a modeling solution: changes to the feature engineering. In Figure 1.9, we see two data scientists attempting to make a set of changes in a monolithic code base. In this development paradigm, all of the logic for the entire job is written in a single notebook through scripted variable declarations and functions.

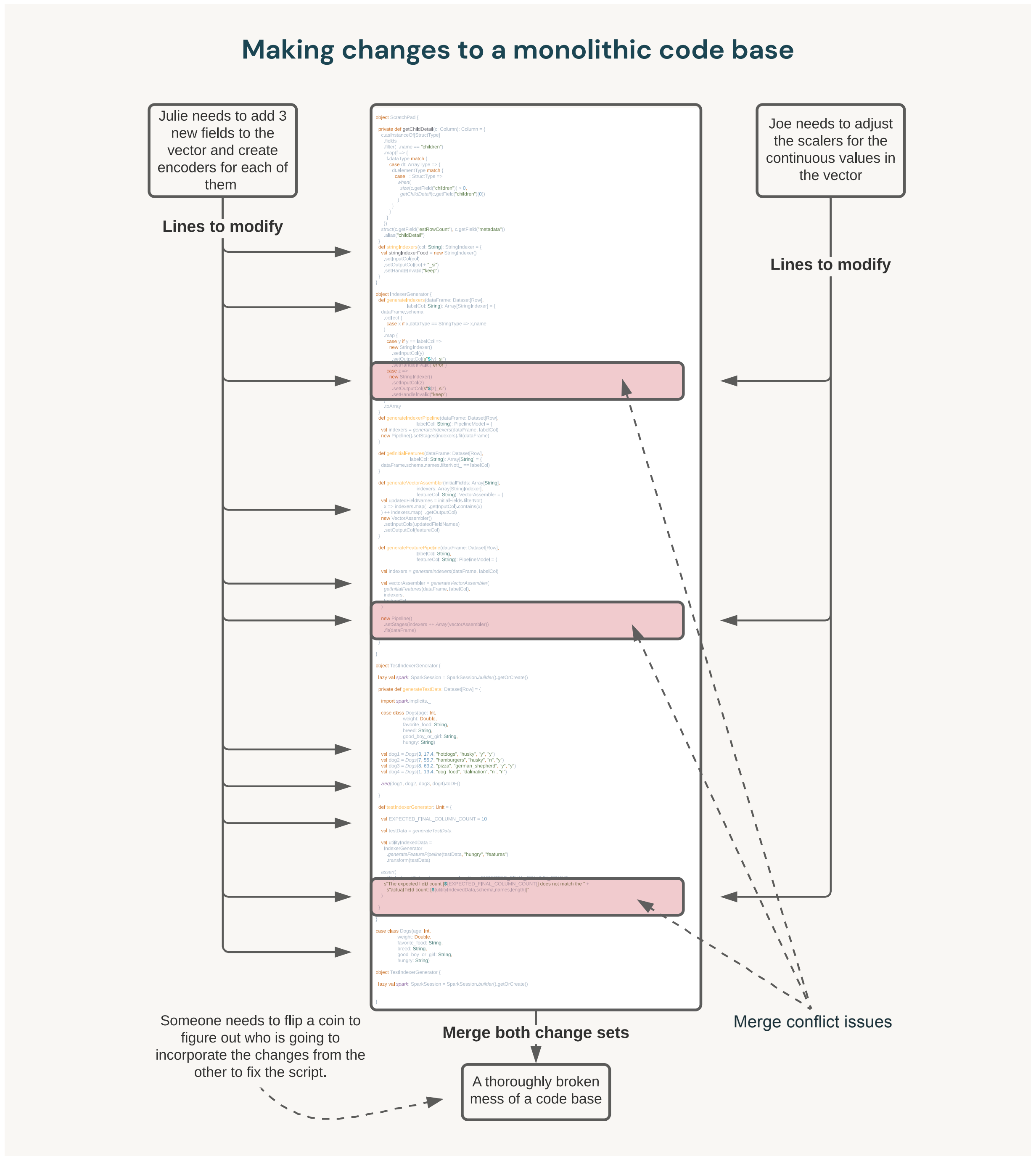

Making changes to a monolithic code base
Julie needs to add 3 new fields to the vector and create encoders for each of them
Lines to modify
Joe needs to adjust the scalers for the continuous values in the vector
Lines to modify
Someone needs to flip a coin to figure out who is going to incorporate the changes from the other to fix the script.
Merge both change sets
Merge conflict issues
A thoroughly broken mess of a code base

Figure 1.9
Editing a monolithic code base (a script) for ML project work

Julie, working in the monolithic code base, is likely to have a lot of searching and scrolling to do, finding each individual location where the feature vector is defined and adding her new fields to collections. Her encoding work will need to be correct and carried throughout the script in the correct places as well. It's a daunting amount of work for any sufficiently complex ML code base (where the number of lines of code for feature engineering and modeling combined can reach to the thousands if developed in a scripting paradigm) and is prone to frustrating errors in the form of omissions, typos and other transcription errors.

Joe, meanwhile, has far fewer edits to do, but is still subject to the act of searching through the long code base and relying on editing the hard-coded values correctly.

The real problem with the monolithic approach comes when they try to incorporate each of their changes into a single copy of the script. As they both have mutual dependencies on one another's work, they will both have to update their code and select one of their copies to serve as a "master" for the project, copying the changes from the other's work. It will be a long and arduous process, wasting precious development time and likely requiring a great deal of debugging to get correct.

Figure 1.10 shows a different approach to maintaining an ML project's code base, utilizing modularized code architecture to separate the tight coupling that is present within the large script from Figure 1.9.

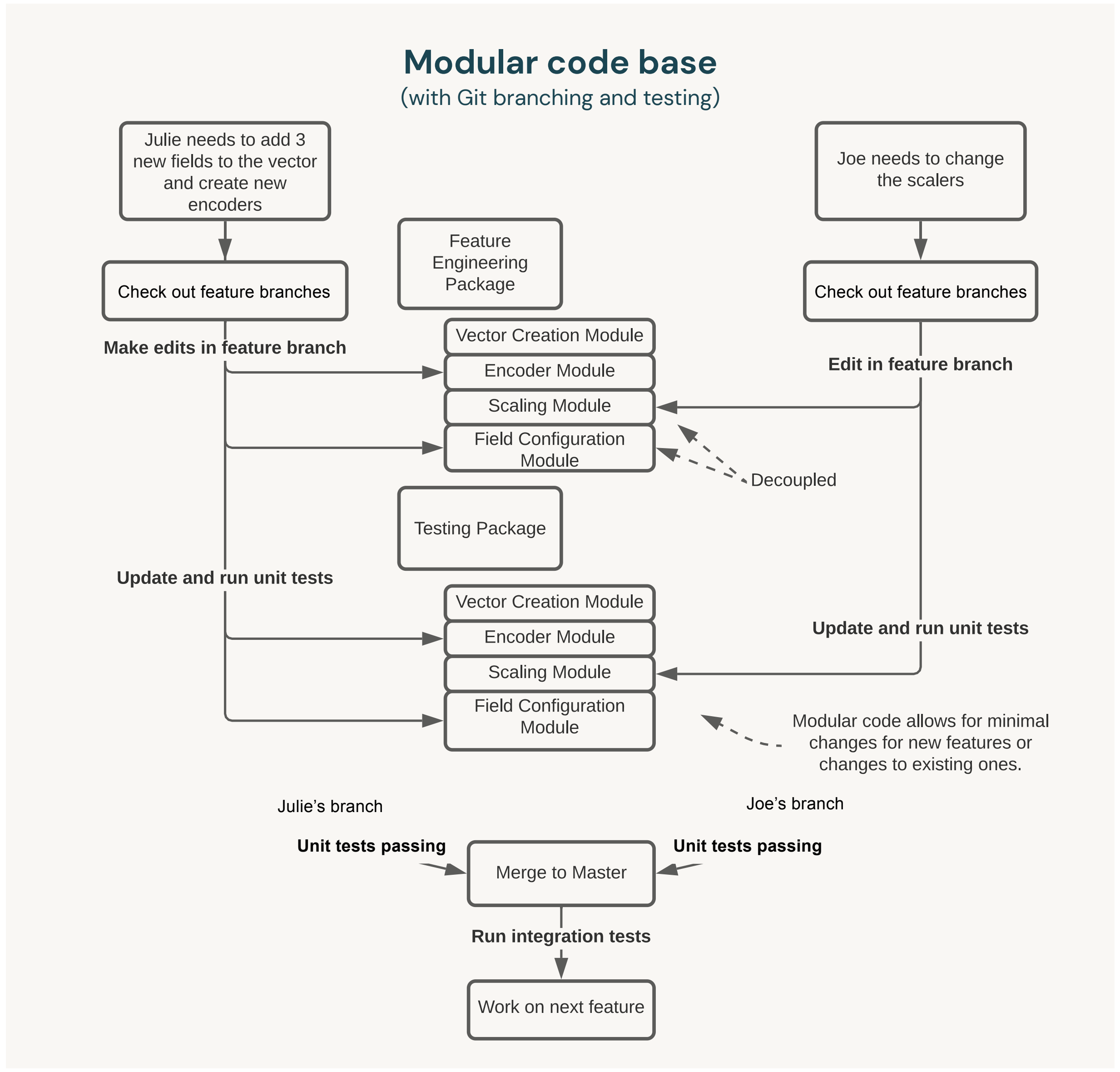

Modular code base
(with Git branching and testing)
Julie needs to add 3 new fields to the vector and create new encoders
Check out feature branches
Make edits in feature branch
Feature Engineering Package
Vector Creation Module
Encoder Module
Scaling Module
Field Configuration Module
Joe needs to change the scalers
Check out feature branches
Edit in feature branch
Decoupled
Testing Package
Update and run unit tests
Vector Creation Module
Encoder Module
Scaling Module
Field Configuration Module
Update and run unit tests
Modular code allows for minimal changes for new features or changes to existing ones.
Julie's branch
Joe's branch
Unit tests passing
Merge to Master
Unit tests passing
Run integration tests
Work on next feature

Figure 1.10
Updating of a modular ML code base to prevent rework and merge conflicts

Figure 1.10 shows a modularized code base written in an integrated development environment (IDE). While both changes that are being made by the two data scientists are identical in their nature to those being made in Figure 1.9 (Julie is adding a few fields to the feature vector and updating encodings for these new fields, while Joe is updating the scaler used on the feature vector), the amount of effort and time spent getting these changes working in concert with one another is dramatically different.

With a fully modularized code base registered in Git, both of them can check out a feature branch from the master, make their small edits to the modules that are part of their features, write some new tests (if needed), run their tests and submit a pull request. Once their work is complete, due to the configuration-based code and the fact that the methods in each of the module classes can act upon the data for their project through leveraging the job configuration, each of their feature branches will not impact one another and should just work as designed. They can cut a release branch of both of their changes in a single build, run a full integration test and safely merge to the master, confident in the fact that their work is correct.

Writing code in this manner (modular design, written in an IDE) is a large departure for many data scientists. We've learned in interactive notebooks, and many of us (myself included) still use notebooks quite frequently for prototyping ideas, for experimentation and for analysis of our work. However, by adopting this alternate way of writing ML code (porting prototype scripts and functions into object-oriented or functional programming paradigms), projects can support many users simultaneously developing new features for them, as well as ensure that each new idea and bit of functionality is tested fully to eliminate bugs that are difficult to track down. The overhead in time and effort associated with creating an ML code framework based in these long-ago proven paradigms of software development will be thoroughly worth it once even the second change to the code base needs to be done.

DEPLOYMENT

Perhaps the most confusing and complex part of ML project work comes at the point long after a powerfully accurate model is built. The path between the model creation and the serving of the predictions to a point that they can be used is nearly as difficult, and its possible implementations nearly as varied, as there are models to serve prediction needs.

Let's take a company that provides analysis services to the fast food industry as an example for this section. They've been fairly successful in serving predictions for inventory management at region-level groupings for years, running large batch predictions for the per day demands of expected customer counts at a weekly level, submitting their forecasts as bulk extracts each week.

The data science team up until this point has been accustomed to an ML architecture that effectively looks like Figure 1.11.

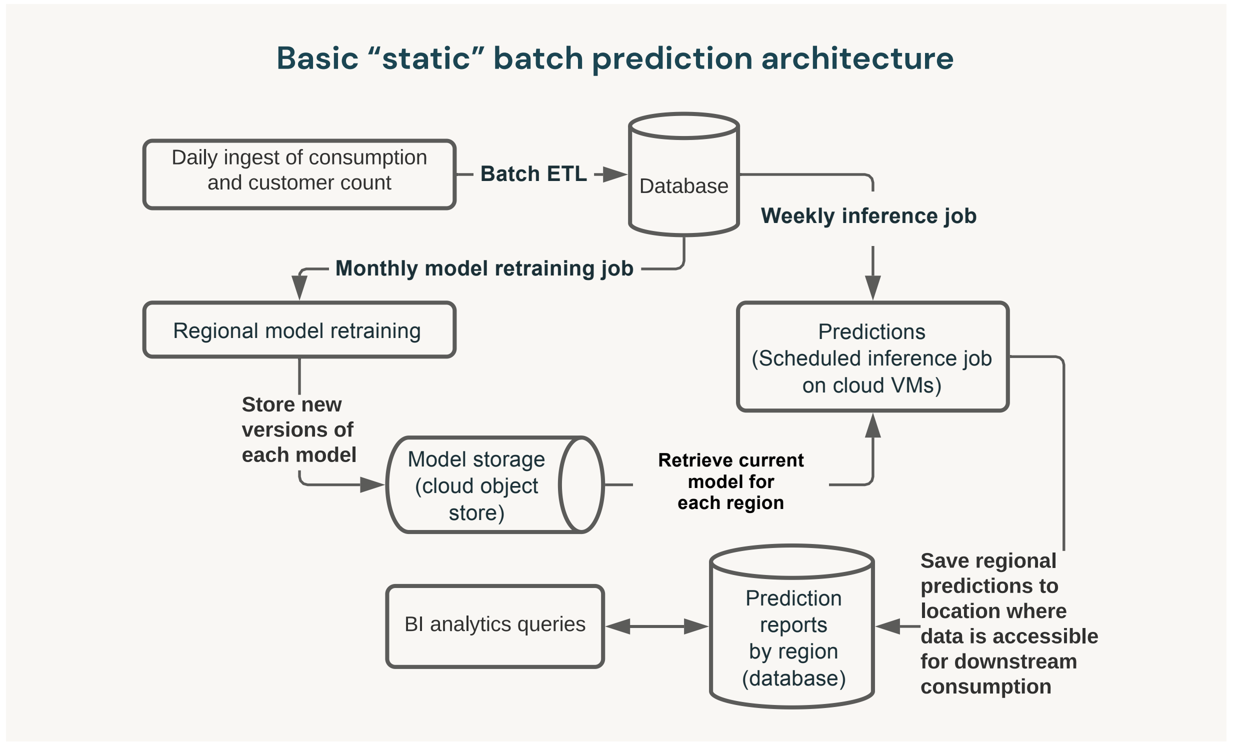

Basic "static" batch prediction architecture
Daily ingest of consumption and customer count
Batch ETL
Database
Weekly inference job
Monthly model retraining job
Regional model retraining
Predictions (Scheduled inference job on cloud VMs)
Store new versions of each model
Model storage (cloud object store)
Retrieve current model for each region
Save regional predictions to location where data is accessible for downstream consumption
BI analytics queries
Prediction reports by region (database)

Figure 1.11
The relatively simple scheduled batch internal-facing prediction serving architecture

This relatively standard architecture for serving up scheduled batch predictions (shown in Figure 1.11), solely focused on exposing inference results to internal analytics personnel, isn't particularly complex and is a paradigm that they are very familiar with. With the scheduled synchronous nature of the design, as well as the large amounts of time between subsequent retraining and inference, the general sophistication of their technology stack doesn't have to be particularly high (which is a good thing; see the note below).

A brief note on simplistic architecture

In the world of ML, one should always strive to use the simplest design possible when building an architecture. If the project requires a periodicity of inference of 1 week, then use a batch process (not real-time streaming). If the data volumes are in the megabytes, then use a database and a simple VM (not a 25-node Spark cluster). If the runtime of training is measured in minutes, stick to CPUs (not GPUs).

Using complex architecture, platforms and technology simply for the sake of using them will create a condition that you will inevitably regret, as it introduces unnecessary complexity to an already complex solution. With each new complexity that is introduced, the chances rise that something is going to break (usually in a spectacularly complex manner). Keeping the technology, the stack and the architecture as simple as is needed to solve the imminent business needs of the project is always a recommended best practice for delivering a consistent, reliable and effective solution to a business unit.

As the company has realized the benefits of predictive modeling over time, when a new business segment opens up, the business unit approaches the data science team to build a new prediction system for them. This new service is one requiring an approach to inventory forecasting at a per store level, with a requirement that the predictions respond in near real-time throughout the day. Realizing that they need to do more than just build a completely different ensemble of models to solve this use case, the data science team focuses most of their time and energy on the ML portion of the project. They don't realize that not only does the serving component of this solution need to rely on a REST API to serve the data to individual store owners through an application, but also that they would have to be frequently updating the per store forecasts fairly frequently throughout the day.

After coming up with an architecture that supports the business need (months after the start of the project, well after the modeling portion of the project had been finished), they proceed to build it with the assistance of some Java software engineers. It wasn't until after the first week of going live that the business realized that the costs of implementing this in the cloud are more than an order of magnitude higher than the revenue they are getting for the service. The new architecture that is needed to serve the business need is shown in Figure 1.12.

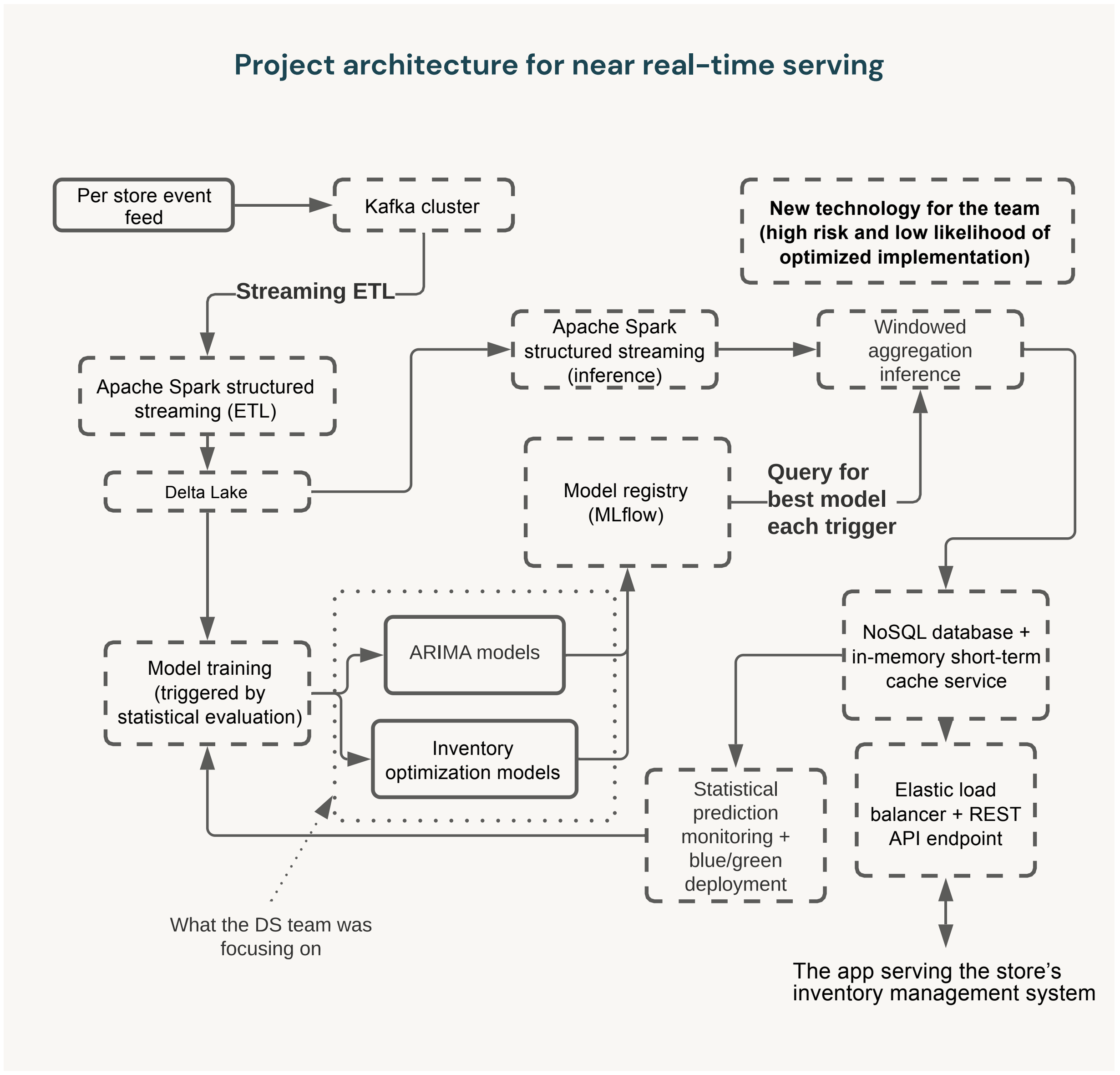

Project architecture for near real-time serving
Per store event feed
Kafka cluster
New technology for the team (high risk and low likelihood of optimized implementation)
Streaming ETL
Apache Spark structured streaming (ETL)
Apache Spark structured streaming (inference)
Windowed aggregation inference
Delta Lake
Model registry (MLflow)
Query for best model each trigger
Model training (triggered by statistical evaluation)
ARIMA models
Inventory optimization models
NoSQL database + in-memory short-term cache service
Statistical prediction monitoring + blue/green deployment
Elastic load balancer + REST API endpoint
What the DS team was focusing on
The app serving the store's inventory management system

Figure 1.12
The far more complex pseudo real-time serving architecture required to meet the business needs for the project

It doesn't take long for the project to get canceled and a complete redesign of the architecture and modeling approach to be commissioned to keep the costs down.

This is a story that plays out time and again at companies implementing ML to solve new and interesting problems. Without focusing on the deployment and serving, the chances of project success will be rather limited, due not to the fact that it can't be developed but rather that the engineering of the solution could cost far more money than the project brings in.

Thinking of deployment and serving with a skeptical eye focused on how much it's going to cost to run, maintain and monitor are great habits to have — ones that will help to inform not only the development of the solution but also the feasibility of the general idea that the project is intended to implement. After all, there's no sadder death to an ML project than the one that forces a great solution to be turned off because it simply costs too much to run.

Figure 1.13 shows some of (not all, by any stretch of the imagination) the elements to think about with regard to serving prediction results. Notice the focus on relative expense for each section. Having this information analyzed very early in a project can set the expectation for how expensive the project is going to be so that the palatability of the cost of running it can be measured before the cost of development is incurred by the company.

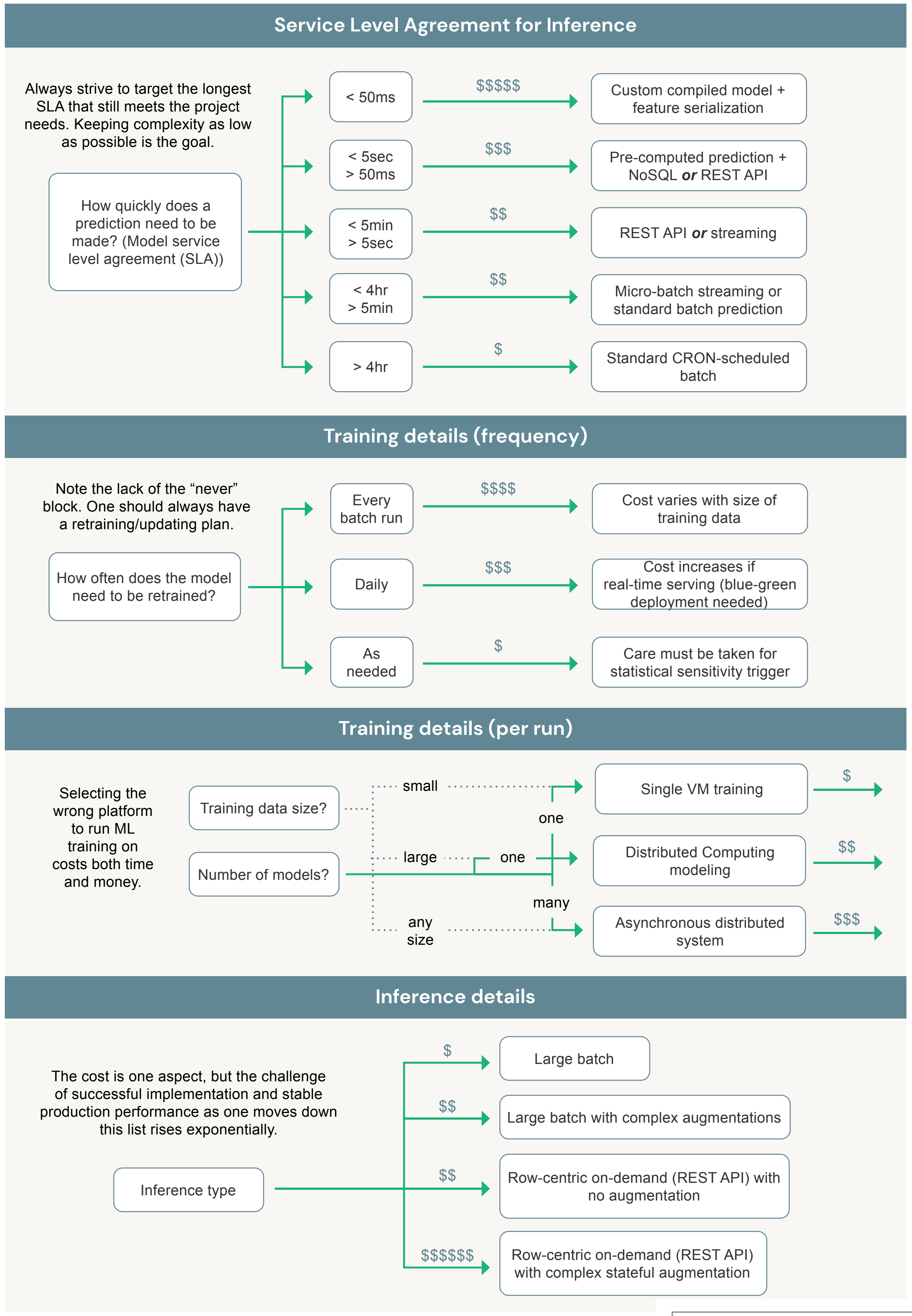

Service Level Agreement for Inference
Always strive to target the longest SLA that still meets the project needs. Keeping complexity as low as possible is the goal.
How quickly does a prediction need to be made? (Model service level agreement (SLA))
< 50ms
$$$$$
Custom compiled model + feature serialization
< 5sec > 50ms
$$$
Pre-computed prediction + NoSQL or REST API
< 5min > 5sec
$$
REST API or streaming
< 4hr > 5min
$$
Micro-batch streaming or standard batch prediction
> 4hr
$
Standard CRON-scheduled batch
Training details (frequency)
Note the lack of the "never" block. One should always have a retraining/updating plan.
How often does the model need to be retrained?
Every batch run
$$$$
Cost varies with size of training data
Daily
$$$
Cost increases if real-time serving (blue-green deployment needed)
As needed
$
Care must be taken for statistical sensitivity trigger
Training details (per run)
Selecting the wrong platform to run ML training on costs both time and money.
Training data size?
Number of models?
small
large
any size
one
one
many
Single VM training
$
Distributed Computing modeling
$$
Asynchronous distributed system
$$$
Inference details
The cost is one aspect, but the challenge of successful implementation and stable production performance as one moves down this list rises exponentially.
Inference type
$
Large batch
$$
Large batch with complex augmentations
$$
Row-centric on-demand (REST API) with no augmentation
$$$$$$
Row-centric on-demand (REST API) with complex stateful augmentation

Figure 1.13
Deployment considerations with respect to both hardware (platform) cost and human capital (time to develop, difficulty in maintaining) for different ML serving paradigms

The full book, “Machine Learning Engineering in Action,” by Keepgoing.AI Publications covers each of the sections from Figure 1.13 (and many others that affect the cost of ML), the considerations for serving, platform selection, build vs. buy, and data volume costs associated with modeling. It’s not the most exciting part of ML engineering, nor is it the most frequently thought about (until it’s too late, usually), but it can be the fastest way to get a project canceled, and as such, should be considered quite seriously.

EVALUATION

The absolutely worst way of getting an ML project canceled or abandoned is by budgetary reasons. Typically, if the project has gotten into production to begin with, the up-front costs associated with developing the solution were accepted and understood by the leadership of the company. Having a project canceled after it’s already in production because of its uncertain impact to the company is a different matter entirely. If you can’t prove the worth of the solution, there’s a very real possibility that someone will tell you to turn it off to save some money someday.

Imagine a company that has spent the past 6 months working tirelessly on a new initiative to increase sales through the use of predictive modeling. They’ve followed best practices throughout the project’s development — making sure that they’re building exactly what the business is asking for, focusing their development efforts on maintainable and extensible code — and have pushed the solution to production. The model has been performing wonderfully over the past 3 months. Each time the team has done post hoc analysis of the predictions to the state of reality afterward, the predictions turn out to be eerily close.

The scenario in Figure 1.14 then rears its ugly head with a simple question from one of the executives at the company who is concerned about the cost of running this ML solution.

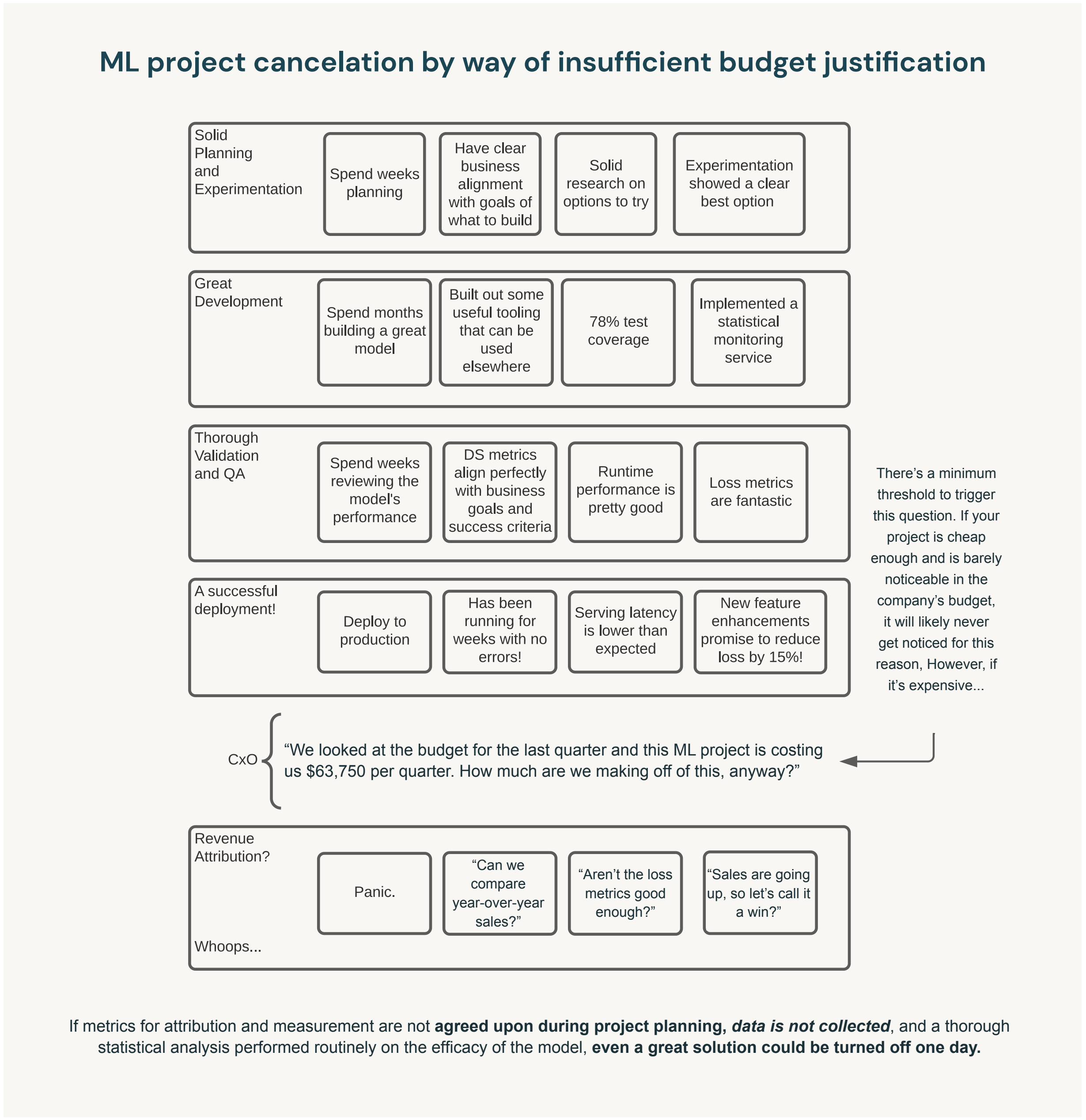

ML project cancelation by way of insufficient budget justification
Solid Planning and Experimentation
Spend weeks planning
Have clear business alignment with goals of what to build
Solid research on options to try
Experimentation showed a clear best option
Great Development
Spend months building a great model
Built out some useful tooling that can be used elsewhere
78% test coverage
Implemented a statistical monitoring service
Thorough Validation and QA
Spend weeks reviewing the model's performance
DS metrics align perfectly with business goals and success criteria
Runtime performance is pretty good
Loss metrics are fantastic
A successful deployment!
Deploy to production
Has been running for weeks with no errors!
Serving latency is lower than expected
New feature enhancements promise to reduce loss by 15%!
There's a minimum threshold to trigger this question. If your project is cheap enough and is barely noticeable in the company's budget, it will likely never get noticed for this reason, However, if it's expensive...
CxO
“We looked at the budget for the last quarter and this ML project is costing us $63,750 per quarter. How much are we making off of this, anyway?”
Revenue Attribution?
Whoops...
Panic.
“Can we compare year-over-year sales?”
“Aren't the loss metrics good enough?”
“Sales are going up, so let's call it a win?”
If metrics for attribution and measurement are not agreed upon during project planning, data is not collected, and a thorough statistical analysis performed routinely on the efficacy of the model, even a great solution could be turned off one day.

Figure 1.14
A nearly flawless ML project getting canceled due to a lack of A/B testing and statistically valid attribution measurement

The one thing that the team forgot when creating a great ML project: How to tie their predictions to some aspect of the business that can justify its existence, as shown in Figure 1.14. The model that they've been working on and that is currently running in production was designed to increase revenue, but when scrutinized for the cost of using it, the team realized that they hadn't thought of an attribution analytics methodology to prove the worth of the solution. Can they simply add up the sales and attribute it all to the model? No, that wouldn't be even remotely correct. Could they look at the comparison of sales to last year's? That wouldn't be correct either, as there are far too many latent factors impacting the sales.

The only thing that they can do to give attribution to their model is to perform A/B testing and use sound statistical models to arrive at a revenue lift (with estimation errors) calculation to show the extent to which additional sales are due to their model. However, the ship has already sailed, as the model has been fully deployed for all customers. The team lost their chance at justifying the continued existence of the model. While it might not be shut off immediately, it certainly will be on the chopping block if the company needs to reduce its budgetary spend.

It's always a good idea to think ahead and plan for this case. Whether it's happened to you yet or not, I can assure you that at some point it most certainly will. It is far easier to defend your work if you have the ammunition at the ready of validated and statistically significant tests showing the justification for the model's continued existence.

SECTION 1: ML ENGINEERING AS A CONCEPT

1.3 The goals of ML engineering

In the most elemental sense, the primary goal of any data scientist is to use statistics, algorithms and predictive modeling to solve a difficult problem that is either too onerous, too monotonous, too error prone or too complex for a human to do. It's not to build the fanciest model, to create the most impressive research paper about their approach to a solution or to search out the most exciting new tech to force into their project work.

The first and foremost goal of applying software engineering fundamentals — DevOps — to the world of ML is the very reason why DevOps was created. It's to increase the chances of having project work be efficient and making it easier to manage incredibly complex code bases. It's to attempt to eliminate the chances of projects getting canceled, code being abandoned and solutions failing to see the light of day.

We're all here in this profession to solve problems. We have a great many tools to play with, a lot of options to consider and an overwhelmingly complex array of knowledge that we need to attain and maintain in order to be effective at what we do. It's incredibly easy to get lost in this overwhelming avalanche of complexity and detail, only for our project work to suffer a cancelation (because of budget, time or complexity), abandonment (from code that is unmaintainable, fragile or both) or re-prioritization (because of poor objectives, unstable predictions or lack of business need). These are all preventable through the judicious application of a core set of rules.

By focusing on the core aspects of project work that have been highlighted in Section 1.2 and that are covered in greater detail throughout this eBook, you can get to the true desired state of ML work: seeing your models run in production and having them solve a true business problem. The goal in the use of these methodologies, technologies and design patterns is to help focus your time and energy on solving the problems that you were hired to solve so that you can move on to solving more of them, making your company and yourself more successful with all the benefits that predictive modeling has to offer. It's to reduce those high failure and abandonment rates so that your future work can be something other than focusing on apologies, rework and constant maintenance of ill-conceived solutions.

You can do this

There is an entire industry out there that is designed to convince you that you can't. That you need to hire them to do all this complex work for you. They make a great deal of money doing this.

You can learn these core concepts and build a team that follows a methodology to approach ML work that can dramatically increase the success rate of project work. It may seem complex and rather confusing at first, but following these guidelines and using the right tooling to help manage the complexity can help any team develop quite sophisticated ML solutions that won't require massive budgets or consume all the free time that a data science team has in "keeping the lights on" for poorly implemented solutions.

You got this.

Before working into the finer details of each of these methodologies and approaches for ML engineering work, see the outline detailed in Figure 1.15. This is effectively a process flow plan for production ML work. Your own work may look either shockingly similar to this or significantly less complex than what is shown. The intent of showcasing this here is to introduce the concept of topics.

As mentioned before, this is the proven way to ensure that your ML project work will actually meet that base goal that we all strive for: to make something useful that isn't a nightmare to maintain; to use the art of data, the science of mathematical algorithms and our limitless creativity to solve problems.

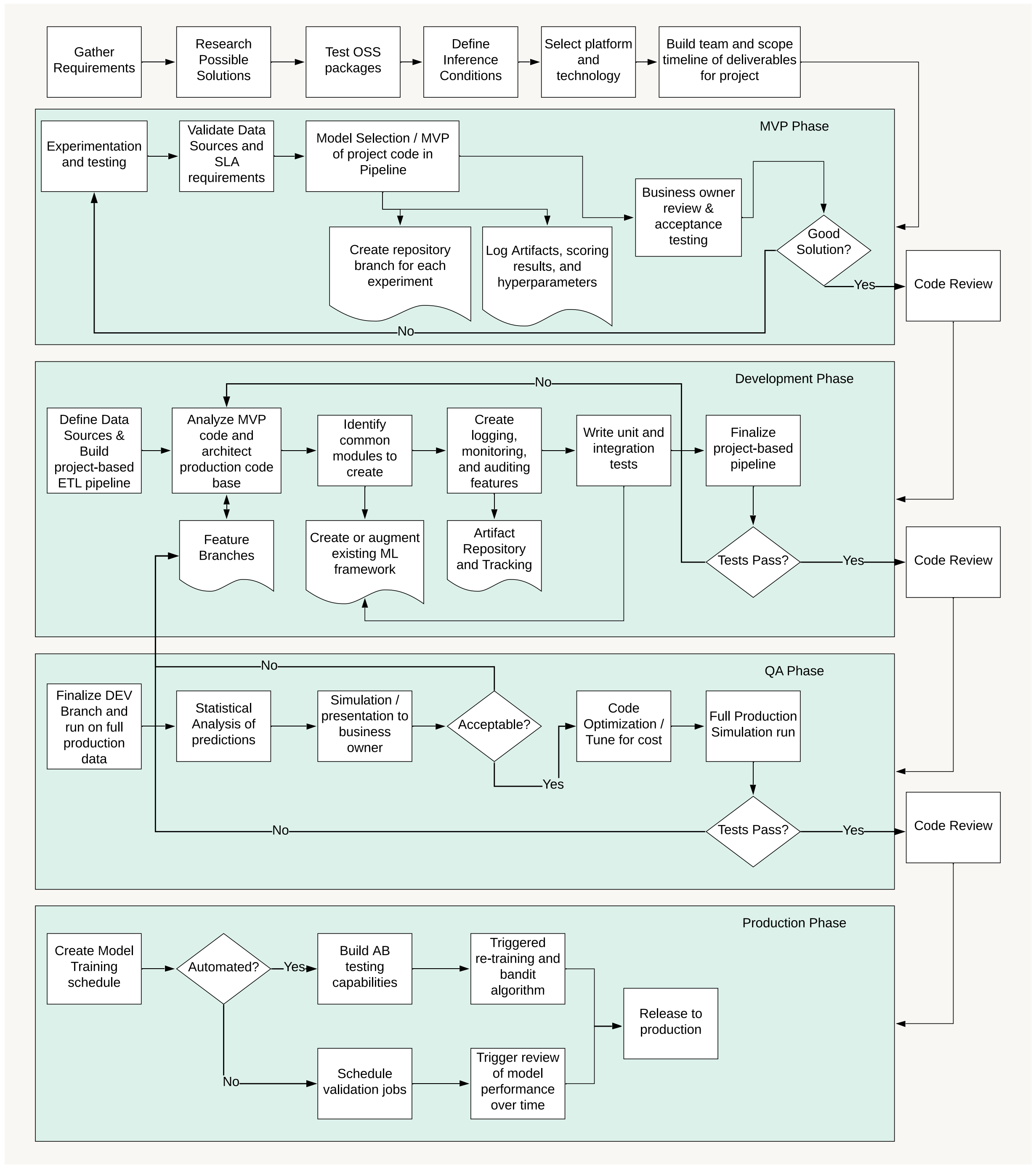

Gather Requirements
Research Possible Solutions
Test OSS packages
Define Inference Conditions
Select platform and technology
Build team and scope timeline of deliverables for project
MVP Phase
Experimentation and testing
Validate Data Sources and SLA requirements
Model Selection / MVP of project code in Pipeline
Business owner review & acceptance testing
Good Solution?
Create repository branch for each experiment
Log Artifacts, scoring results, and hyperparameters
Yes
No
Code Review
Development Phase
No
Define Data Sources & Build project-based ETL pipeline
Analyze MVP code and architect production code base
Identify common modules to create
Create logging, monitoring, and auditing features
Write unit and integration tests
Finalize project-based pipeline
Feature Branches
Create or augment existing ML framework
Artifact Repository and Tracking
Tests Pass?
Yes
Code Review
QA Phase
No
Finalize DEV Branch and run on full production data
Statistical Analysis of predictions
Simulation / presentation to business owner
Acceptable?
Yes
Code Optimization / Tune for cost
Full Production Simulation run
No
Tests Pass?
Yes
Code Review
Production Phase
Create Model Training schedule
Automated?
Yes
Build AB testing capabilities
Triggered re-training and bandit algorithm
Release to production
No
Schedule validation jobs
Trigger review of model performance over time

Figure 1.15
The ML engineering methodology component map. Also, a visual table of contents for this eBook.

SECTION 1: ML ENGINEERING AS A CONCEPT

1.4 Summary

The leading cause of the high rates of ML project work failure in industry is the failure to follow (or having an ignorance of) the six tenets of ML engineering.

ML engineering is a guide — both a toolbox and a map that can help guide project work involving data scientists to ensure that their work adheres to good engineering principles and is focused on solving a business need.

We've seen the core components at a very high level. In Section 2, we delve quite deeply into each of these topics, through the use of active examples, project solutions and simulations of decisions to help give you the tools that you need to build successful, maintainable and resilient code bases that employ machine learning.

SECTION

02

Your data science could use some engineering

keepgoing.ai

SECTION 2

Your data science could use some engineering

In Section 1, we covered the justification for ML engineering, focusing on what a professional and successful approach to data science (DS) work looks like. This section focuses on the "why." The continuing maturity of data science in companies around the world has not only established fundamental methodologies in approaching data science project work but has also created a veritable frenzy at companies, as they get started applying ML to help solve problems and gain a revenue edge.

The unfortunate reality of the situation is that, due to the sudden industry need for skilled engineers versed in ML, there simply aren't enough experienced people (those who have learned lessons the hard way by doing the job for many years) to fill the overwhelming demand. Add to this the extremely deep and complex nature of data science work, which requires many years to become proficient at, and many companies find that more often than not, their project work with ML leads to a great deal of frustration, disappointment and, in the most extreme cases, abandoned projects.

In this section, we define the ecosystem of ML engineering and explain the core reasons for the standards that have been developed and then get into more detail about them.

ML engineering (also known as "MLOps," a term adapted from "DevOps," itself a term for an Agile-inspired collection of tools, methodology and processes for general software development) is not intended to be a specific job title. Nor is it a contradictory approach to data science work. It is not a realm of responsibility wholly divorced from data science solutions either. Rather, it is a *complementary* (and, arguably, critically necessary) additional set of tools, processes and paradigms for conducting data science work. The end goal of which is the creation of more sustainable solutions to business problems at a much lower total cost of ownership. After all, **the primary focus of all data science work is to *solve problems***. Conforming work patterns to a proven methodology that is focused on maintainability and efficiency translates directly to **solving more problems with much less effort**.

SECTION 2: YOUR DATA SCIENCE COULD USE SOME ENGINEERING

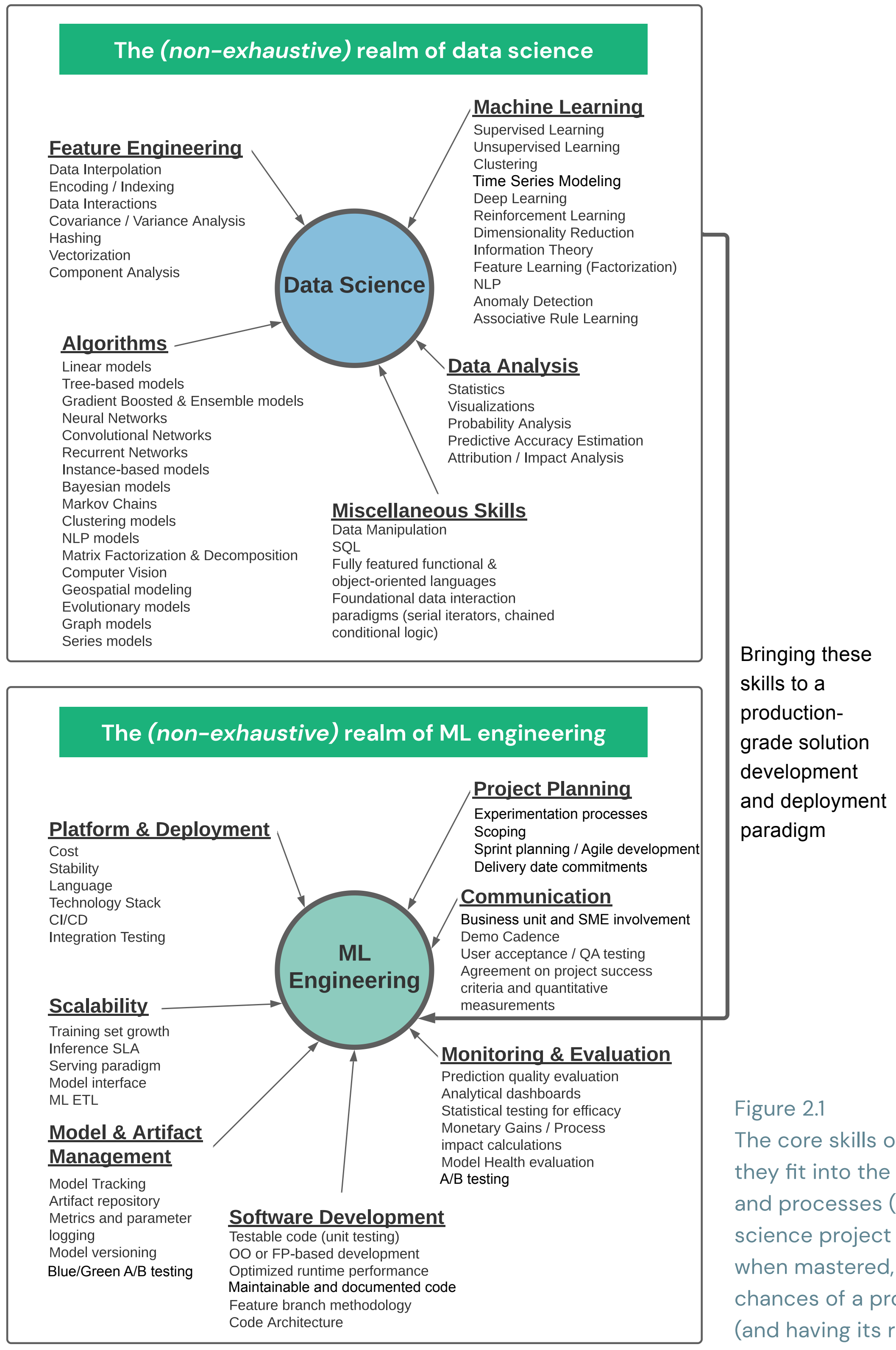
2.1 Augmenting a complex profession with processes that lead to greater success in project work

In one of the earliest uses of the term "data science" — in the 1996 book "Data Science, Classification, and Related Methods," edited by C. Hayashi, C. Yajima, H. H. Bock, N. Ohsumi, Y. Tanaka, and Y. Baba — it is defined as having three areas of focus:

- **Design for data:** Specifically, the planning around how information is to be collected and in what structure it will need to be acquired in order to solve a particular problem
- **Collection of data:** The act of acquiring said data
- **Analysis on data:** Divining insights from the data through the use of statistical methodologies in order to solve a problem

A great deal of modern data science is mostly involved in analysis on data (although there are many cases where a data science team is forced to develop their own ETL), as the first two focus areas are generally handled by a modern data engineering team. Although "analysis on data" is a broad term, most of what a modern data scientist focuses on falls within this category, including applying statistical techniques, data manipulation activities and statistical algorithms (models) to garner insights from and to make predictions on data.

Figure 2.1, in the top portion, illustrates (in an intentionally brief and high-level manner) the focus of the modern data scientist from a technical perspective. These are the elements of the profession that most people focus on when describing what a data scientist does — from data access to building complex predictive models utilizing a dizzying array of algorithmic approaches and advanced statistics. It isn't a particularly accurate assessment of what a data scientist actually does when engaged in project work, but rather focuses on some of the tasks and tools that are employed in solving problems. Thinking of data science in this manner is nearly as unhelpful as classifying the job of a software developer by listing languages, algorithms, frameworks, computational efficiency and other technological considerations of their profession.



In Figure 2.1, we can see how the technological focus of data science (which many practitioners focus on exclusively) outlined in the top portion is but one aspect of the broader system that is shown in the bottom portion. It is in this region — ML engineering — that the complementary tools, processes and paradigms provide a framework of guidance, foundationally supported by the core aspects of data science technology, to work in a more constructive way. ML engineering as a concept is a paradigm that helps practitioners focus on the only aspect of project work that truly matters: addressing problems with solutions that actually work.

Where to start, though?

Figure 2.1
The core skills of data science (above) and how they fit into the broader realm of methods, tools and processes (below) that define successful data science project work. These broader sets of skills, when mastered, can dramatically improve the chances of a project being declared successful (and having its results actually be used).

SECTION 2: YOUR DATA SCIENCE COULD USE SOME ENGINEERING

2.2 A foundation of simplicity

When it comes to truly explaining what a data scientist actually does, nothing can be more succinct than "they solve problems through the creative application of mathematics to data." As broad as that is, it reflects the wide array of solutions that can be developed from recorded information (data). There is nothing forbidden (at least that I'm aware of) regarding expectations of what a data scientist does regarding algorithms, approaches or technology while in the pursuit of solving a business problem. Quite the contrary, as a matter of fact. *Data scientists are problem solvers*, utilizing a wide array of techniques and approaches.

Unfortunately for newcomers to the field, many data scientists believe that they are only providing value to a company when they are using the latest and "greatest" tech that comes along. Instead of focusing on the buzz surrounding some new approach catalogued in a seminal whitepaper or advertised heavily in a blog post, a seasoned data scientist realizes that the only thing that really matters is the act of solving problems, regardless of methodology. As exciting as new technology and approaches are, the effectiveness of a data science team is measured in the quality, stability and cost of a solution that they provide.

As Figure 2.2 shows, one of the most important parts of ML work is navigating the path of complexity when facing any problem. By approaching each new request from a business with this mindset as the veritable cornerstone of ML principles (focusing on the simplest possible solution to the business' problem), the solution itself can be focused on, rather than a particular approach or fancy new algorithm.

Focusing on the principle of pursuing the simplest possible implementation to solve a problem, as illustrated in Figure 2.2, provides the foundation upon which all other aspects of ML engineering are built. It is by far the single most important element of ML engineering, as it will inform all other aspects of project work, scoping and implementation. Striving to "exit the path as early as possible" can be the single biggest driving factor in determining whether a project will fail or not.

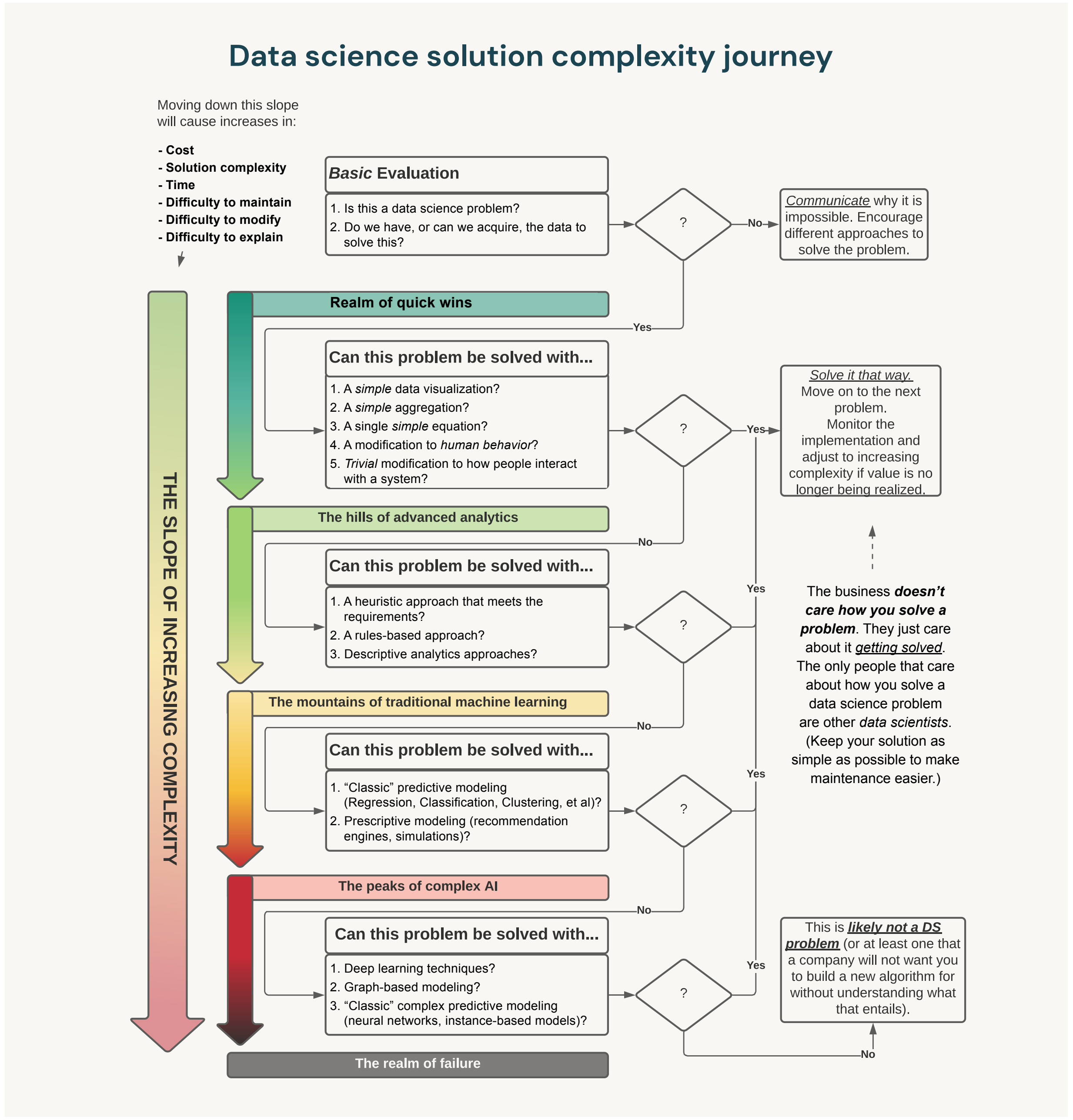

Data science solution complexity journey
Moving down this slope will cause increases in:
- Cost
- Solution complexity
- Time
- Difficulty to maintain
- Difficulty to modify
- Difficulty to explain
THE SLOPE OF INCREASING COMPLEXITY
Basic Evaluation
1. Is this a data science problem?
2. Do we have, or can we acquire, the data to solve this?
?
No
Communicate why it is impossible. Encourage different approaches to solve the problem.
Yes
Realm of quick wins
Can this problem be solved with...
1. A simple data visualization?
2. A simple aggregation?
3. A single simple equation?
4. A modification to human behavior?
5. Trivial modification to how people interact with a system?
Yes
Solve it that way. Move on to the next problem. Monitor the implementation and adjust to increasing complexity if value is no longer being realized.
No
The hills of advanced analytics
Can this problem be solved with...
1. A heuristic approach that meets the requirements?
2. A rules-based approach?
3. Descriptive analytics approaches?
The business doesn't care how you solve a problem. They just care about it getting solved. The only people that care about how you solve a data science problem are other data scientists. (Keep your solution as simple as possible to make maintenance easier.)
No
The mountains of traditional machine learning
Can this problem be solved with...
1. "Classic" predictive modeling (Regression, Classification, Clustering, et al)?
2. Prescriptive modeling (recommendation engines, simulations)?
No
The peaks of complex AI
Can this problem be solved with...
1. Deep learning techniques?
2. Graph-based modeling?
3. "Classic" complex predictive modeling (neural networks, instance-based models)?
This is likely not a DS problem (or at least one that a company will not want you to build a new algorithm for without understanding what that entails).
No
The realm of failure

Figure 2.2
The decision path to minimize complexity in implementing data science projects. From the point of basic evaluation (is this even a data science problem?) to the end (completely impossible to solve with current technology), gating the decisions in increasing order of complexity is the recommended approach for tackling any project work. The simplest solution that can still solve the problem is always going to be the best one. Heading for the most "impressive" solution or the latest fad will always increase the risk that a project will fail to materialize (or be used by the business) due to complexity, cost, time to develop or poor interpretability.

SECTION 2: YOUR DATA SCIENCE COULD USE SOME ENGINEERING

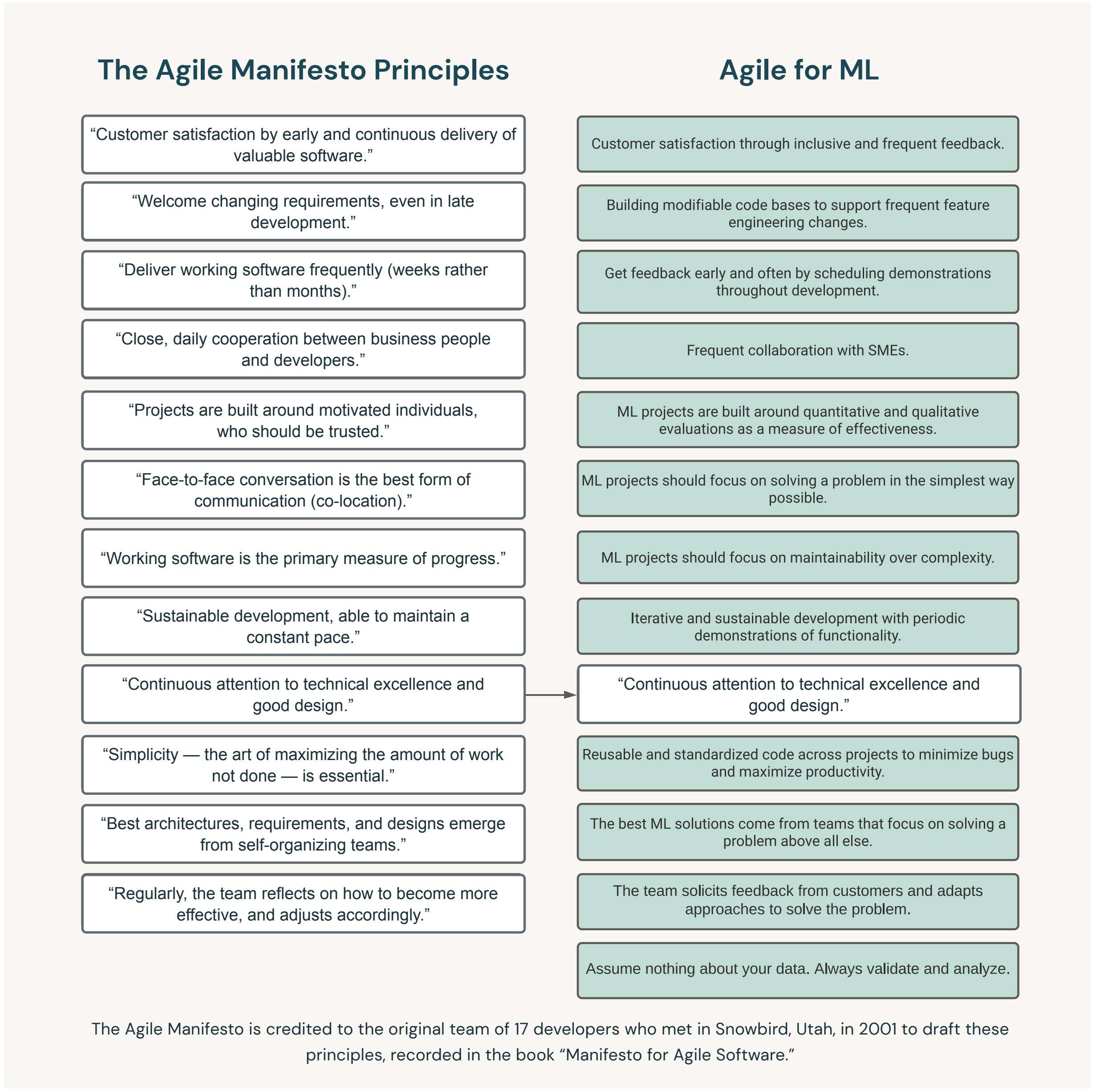
2.3 Co-opting principles of agile software engineering

DevOps brought guidelines and a demonstrable paradigm of successful engineering work to software development. With the advent of the Agile Manifesto, seasoned industry professionals recognized the failings of how software had been developed.

As Figure 2.3 shows, with a slight modification to the principles of agile development, we can come up with a set of guidelines for the application of data science to business problems.

In this eBook, we will cover all these topics, highlight why they are important and give examples of how to apply them to solve business problems. While some are a significant departure from the principles of Agile, the applicability to ML project work has provided repeatable patterns of success for us and many others.

There are, however, two critical points of agile development that can, when applied to ML project work, dramatically improve the way that a data science team approaches their work.



The Agile Manifesto Principles	Agile for ML
"Customer satisfaction by early and continuous delivery of valuable software."	Customer satisfaction through inclusive and frequent feedback.
"Welcome changing requirements, even in late development."	Building modifiable code bases to support frequent feature engineering changes.
"Deliver working software frequently (weeks rather than months)."	Get feedback early and often by scheduling demonstrations throughout development.
"Close, daily cooperation between business people and developers."	Frequent collaboration with SMEs.
"Projects are built around motivated individuals, who should be trusted."	ML projects are built around quantitative and qualitative evaluations as a measure of effectiveness.
"Face-to-face conversation is the best form of communication (co-location)."	ML projects should focus on solving a problem in the simplest way possible.
"Working software is the primary measure of progress."	ML projects should focus on maintainability over complexity.
"Sustainable development, able to maintain a constant pace."	Iterative and sustainable development with periodic demonstrations of functionality.
"Continuous attention to technical excellence and good design."	"Continuous attention to technical excellence and good design."
"Simplicity — the art of maximizing the amount of work not done — is essential."	Reusable and standardized code across projects to minimize bugs and maximize productivity.
"Best architectures, requirements, and designs emerge from self-organizing teams."	The best ML solutions come from teams that focus on solving a problem above all else.
"Regularly, the team reflects on how to become more effective, and adjusts accordingly."	The team solicits feedback from customers and adapts approaches to solve the problem.
	Assume nothing about your data. Always validate and analyze.

The Agile Manifesto is credited to the original team of 17 developers who met in Snowbird, Utah, in 2001 to draft these principles, recorded in the book "Manifesto for Agile Software."

Figure 2.3
Agile Manifesto elements adapted to ML project work

keepgoing.ai

Communication and cooperation

As will be discussed many times throughout this book (particularly in the next two sections), the core tenets of successful ML solution development are focused on people. This may seem incredibly counterintuitive for a profession that is so steeped in mathematics, science, algorithms and clever coding. The reality is that quality implementations of a solution to a problem are never created in a vacuum. The most successful projects are those that focus more on the people and the communications regarding the project and its state rather than on the tools and formal processes (or documentation) surrounding the development of the solution.

In traditional agile development, this rings very true, but for ML work, the interactions between the **people coding the solution and those for whom the solution is being built** are even more critical. This is due to the complexity of what is involved in building the solution. Since the vast majority of ML work is something that is rather foreign to the average layperson, requiring years of dedicated study and continual learning to master, it's especially important to have meaningful and useful discussions.

The single biggest driving factor in making a successful project with the least amount of rework is *collaborative involvement* between the ML team and the business unit. The second biggest factor to ensure success is communication within the ML team.

Approaching project work with a lone-wolf mentality (as has been the focus for most people throughout their academic careers) is counterproductive to solving a difficult problem. Figure 2.4 illustrates this risky behavior.

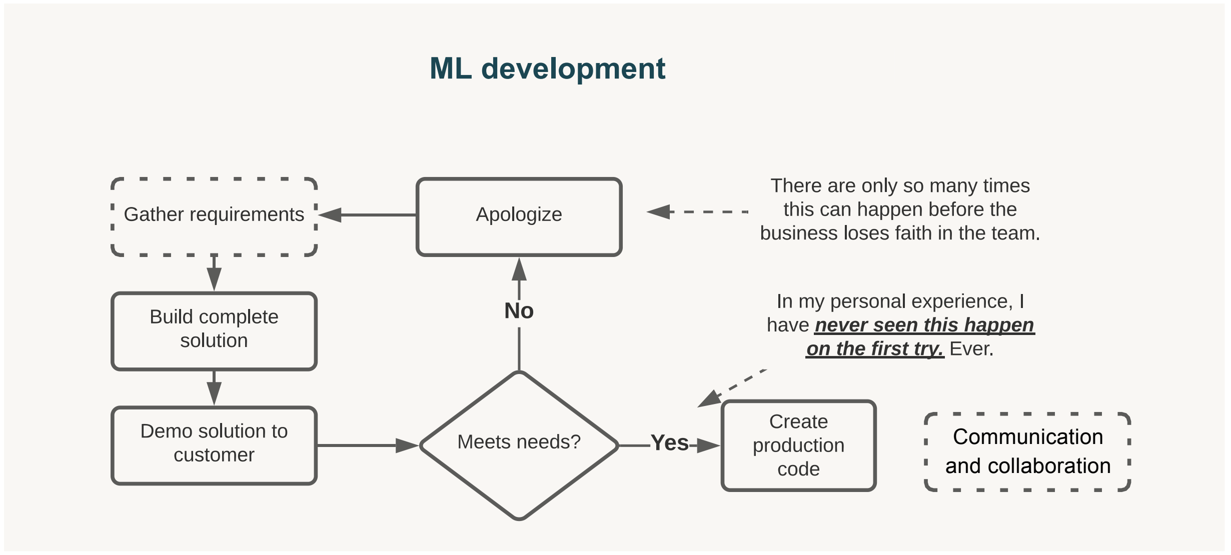

ML development
Gather requirements
Apologize
There are only so many times this can happen before the business loses faith in the team.
Build complete solution
No
In my personal experience, I have ***never seen this happen on the first try.*** Ever.
Demo solution to customer
Meets needs?
Yes
Create production code
Communication and collaboration

Figure 2.4
The hard-learned lesson of working on a full ML solution in isolation. It rarely ends well.

The reasons for the development style in Figure 2.4 can be many, but the end result is typically the same: either a lot of rework or a lot of frustration on the part of the business unit. Even if there aren't any other data science team members (a "team" of a single person), it can be helpful to ask for peer reviews and demonstrate the solution to other software developers, an architect or subject matter experts from the business unit that the solution is being built for. The absolute last thing that you want to do is gather requirements and head off to a keyboard to solve a problem without ever talking to anyone. The chances of meeting all the project requirements, getting the edge cases right and building what the customer is expecting are so infinitesimally small that, should it work out well, perhaps you should look into buying some lottery tickets with all the excess luck that you have to spare.

A more comprehensive and agile-aligned development process for ML bears a close resemblance to agile general software development. The only main difference is that extra levels of internal demonstrations aren't normally required for software development (a peer review feature branch typically suffices there). For ML work, it's important to show the performance as a function of how it affects the data being passed into your code, demonstrate functionality and show visualizations of the output. Figure 2.5 shows a preferable agile-based approach to ML work, focused heavily on collaboration and communication both internally and externally.

The greater level of interaction between team members will nearly always contribute to more ideas, perspectives and challenges to assumed facts, leading to a higher-quality solution. If you leave either your customers (the business unit requesting your help) or your peers out of the discussions (even around minute details in development choices), the chances that you'll build something they weren't expecting — or desiring — will go up.

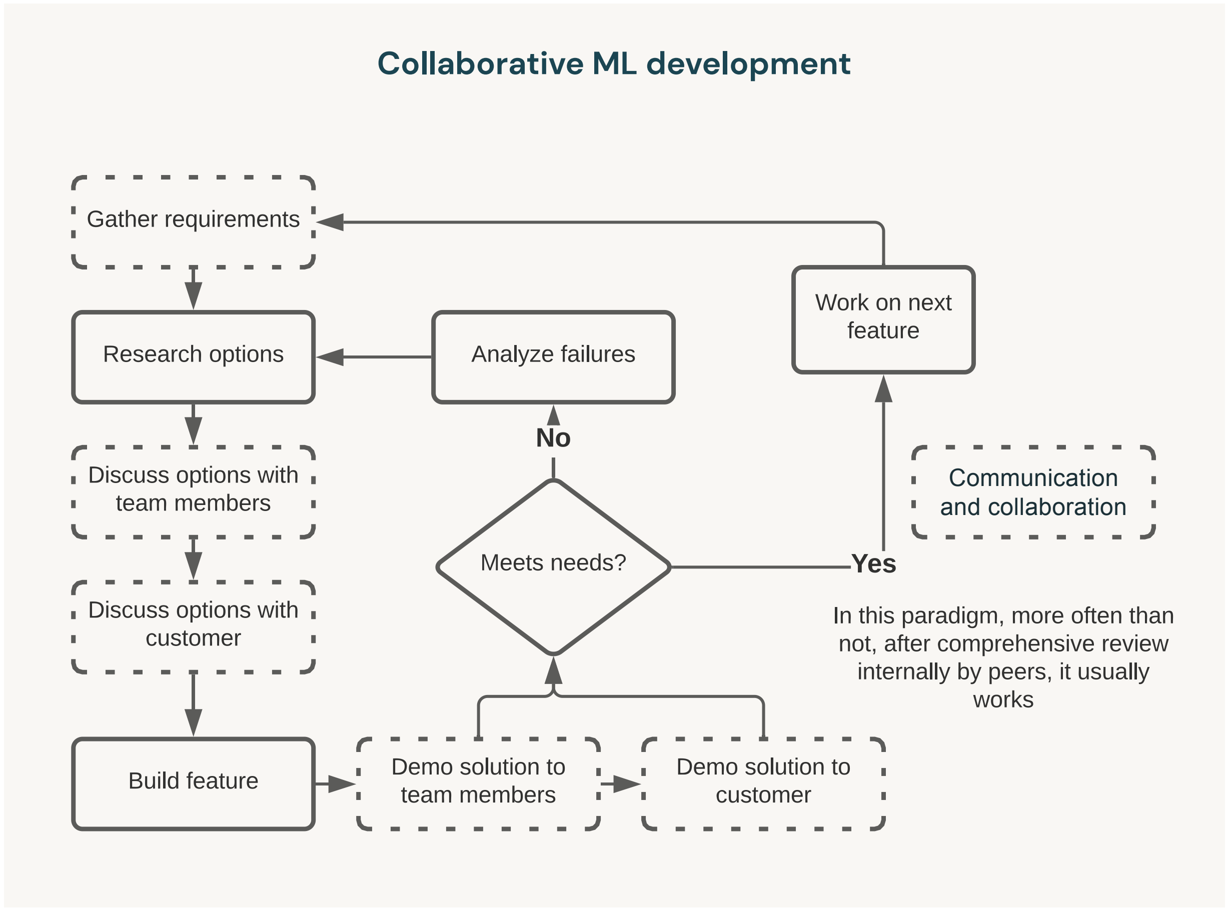

Collaborative ML development
Gather requirements
Research options
Analyze failures
Work on next feature
No
Discuss options with team members
Communication and collaboration
Meets needs?
Yes
Discuss options with customer
In this paradigm, more often than not, after comprehensive review internally by peers, it usually works
Build feature
Demo solution to team members
Demo solution to customer

Figure 2.5
Agile-based ML project development focused heavily on constant communication, open feedback and collaboration. This approach to project work has proven to be very successful.

keepgoing.ai

Embracing and expecting change

It is of utmost importance to be prepared for changes and even to consider them inevitable — not only with regard to experimentation and the direction of a project but also when it comes to development. In nearly every ML project, the things that were defined as goals at the beginning were never exactly what was built by the end of the project.

This applies to everything from specific technologies, development languages and algorithms to assumptions or expectations about the data, and sometimes even to the usage of ML to solve the problem in the first place (a simple aggregation dashboard to help people solve a problem more efficiently, for example).

If you plan for the inevitable change, you can stay focused on *what is most important in all data science work: **solving problems***. It can also keep you from getting distracted by insignificant elements (which fancy algorithm, cool new technology or amazingly powerful framework to develop a solution in).

Without expecting or allowing for change to happen, you might make decisions about a project's implementation that could be incredibly challenging (or impossible) to modify without a full rewrite of all the work you've done up to that point. By thinking about how the direction of the project could change, the work is forced more into a modular format of loosely coupled pieces of functionality, reducing the impact of a directional pivot on other parts of the already completed work.

Agile embraces this concept of loosely coupled design and a very strong focus on building new functionality in iterative sprints where it, even in the face of dynamic and changing requirements, continues to function. By applying this paradigm to ML work, abrupt and even late changes can be made relatively simply (within reason, of course — moving from a tree-based algorithm to a deep learning algorithm is not something that can happen in a two-week sprint). While *simplified*, this **doesn't guarantee simplicity**, though. The fact still stands that anticipating change and building a project architecture that supports rapid iteration and modification will make the development process much easier.

SECTION 2: YOUR DATA SCIENCE COULD USE SOME ENGINEERING

2.4 The foundation of ML engineering

Now that we've seen the bedrock tenets of data science by adapting Agile principles to ML, let's take a brief look at the entire ecosystem of this system of project work — one that's proven to be successful across my many encounters in industry with building resilient and useful solutions to problems.

As mentioned earlier, the idea of ML engineering (MLOps) as a paradigm is rooted in the application of similar principles to those of DevOps in software development. Figure 2.6 shows what the core functionality of DevOps is.

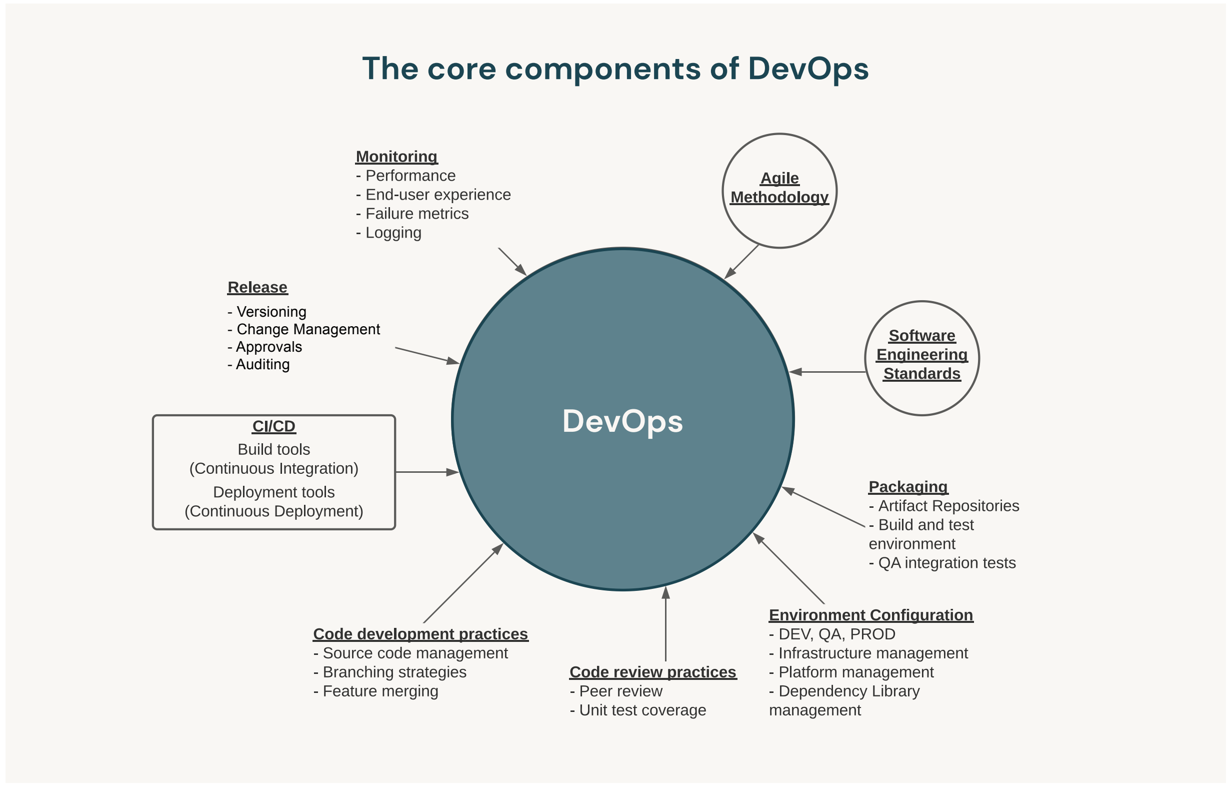


Figure 2.6
The components of DevOps

Comparing these core principles, as we did in section 2.3 to Agile, Figure 2.7 shows the "data science version" of DevOps — MLOps. Through the merging and integration of each of these elements, the most catastrophic events in data science work can be completely avoided: the elimination of failed, canceled or non-adopted solutions.

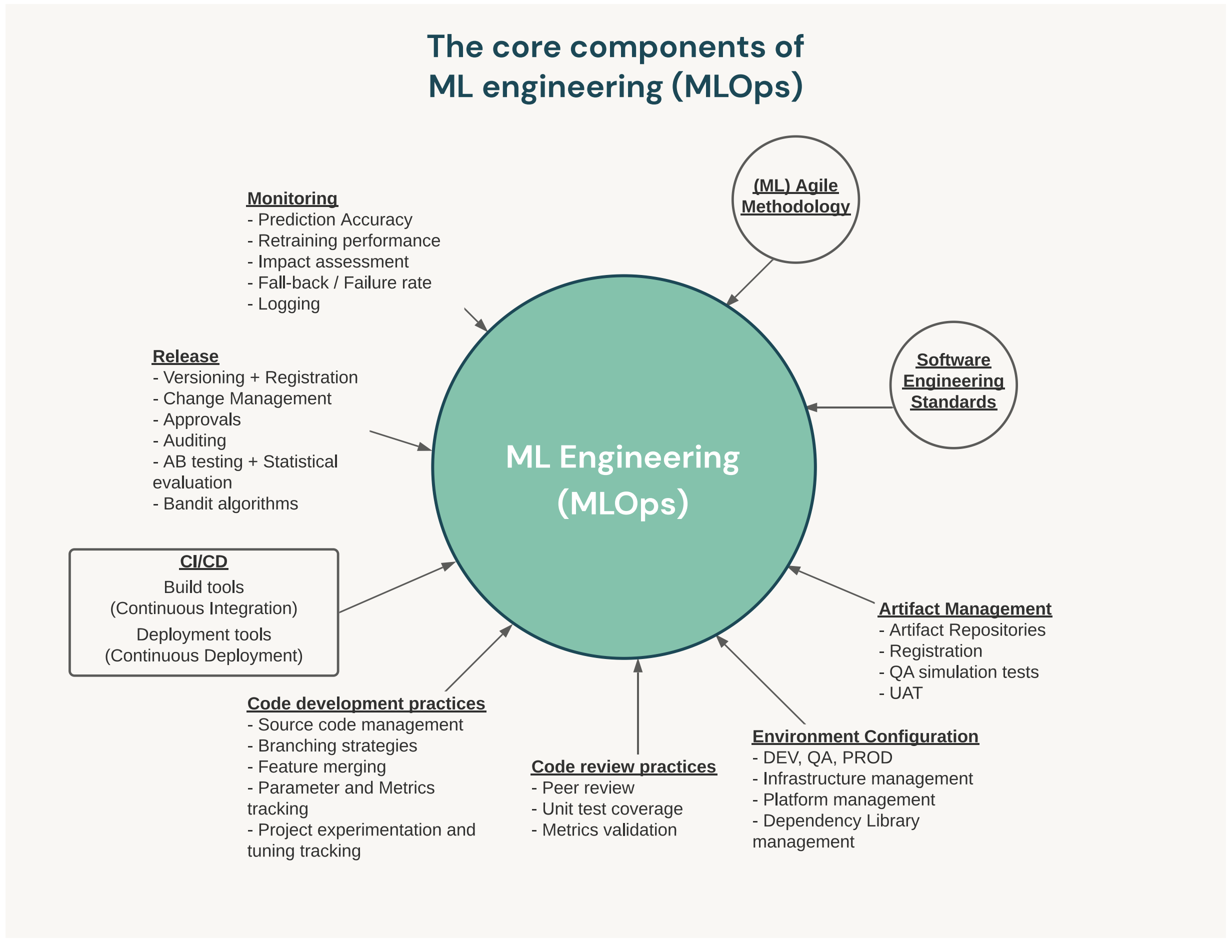


Figure 2.7
Adaptation of DevOps principles to ML project work (MLOps)

keepgoing.ai

SECTION 2: YOUR DATA SCIENCE COULD USE SOME ENGINEERING

2.5 Summary

The application of standard processes, tools and methodologies as an augmentation to data science skill sets helps ensure a higher rate of project success.

The goal of any data science project should not be to use specific tooling or algorithms. Rather, striving for the simplest approach to solving a problem should always be the primary goal in all data science projects (even if it is little more than an analytics visualization).

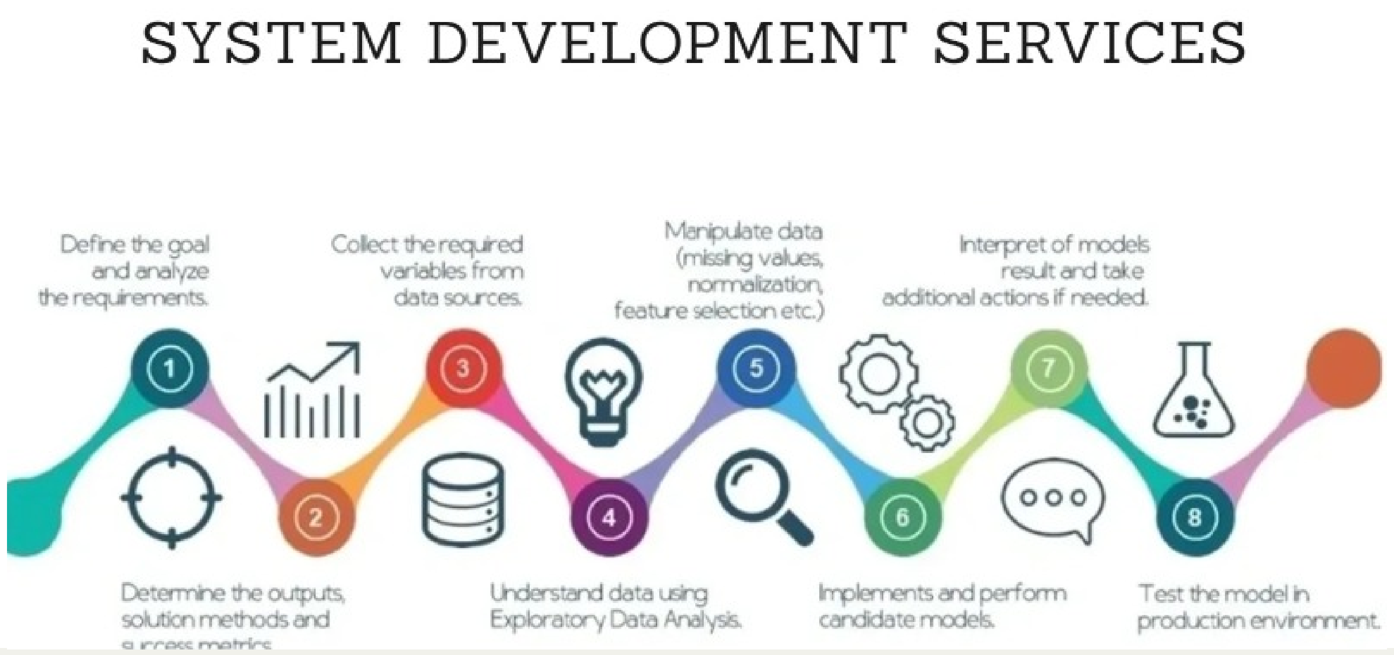
Focusing on an adapted set of principles from agile development can help teams establish patterns of data science work that have been proven successful, ensuring higher quality and more maintainable solutions.

The ecosystem of ML engineering (MLOps) is an adaptation of many of the processes and standards from DevOps, with the addition of specific tooling and domain-specific elements, and created in the pursuit of building resilient, maintainable and production-capable data science solutions.

For a deep dive

If you would like to dive deep into each of the principles explained in this eBook, please visit the Manning website to purchase the full book “Machine Learning Engineering in Action.” Each of these topics will be a focus of chapter-length in-depth discussions in the book, which will not only cover the reasons why each of these elements are important, but also will show useful examples and active implementations that you can follow along with to further cement the practices in your own work. The goal of the book is to make you successful in your machine learning endeavors.

Now that you know what it takes to build production-grade machine learning projects, email to us for more information ai@keepgoing.ai


SYSTEM DEVELOPMENT SERVICES
Define the goal and analyze the requirements.
Collect the required variables from data sources.
Manipulate data (missing values, normalization, feature selection etc.)
Interpret of models result and take additional actions if needed.
1
2
3
4
5
6
7
8
Determine the outputs, solution methods and success metrics
Understand data using Exploratory Data Analysis.
Implements and perform candidate models.
Test the model in production environment.

Feeling ready to get started with real-world machine learning?

Head over to our Solution Accelerators for ready-to-deploy code for your specific industry and use case.

GET STARTED
FREE CONSULTANCY

https://keepgoing.ai/

keepgoing.ai

SYSTEM DEVELOPMENT SERVICES	Hourly Based	€ / Hour	Annual Agreements	€ / Year	Project Based	€ / Project
Machine Learning	✓	69.00	✓	59000.00	✓	4900.00
Deep Learning	✓	99.00	✓	79000.00	✓	6900.00
Data	✓	69.00	✓	59000.00	✓	4900.00
Cloud (Cross)	✓	49.00	✓	49000.00	✓	4900.00
Neuromorphic	✓	199.00	✓	99000.00	✓	9900.00
Prompt	✓	199.00	✓	99000.00	✓	9900.00

EXCLUSIVE NEURAL BOT DEVELOPMENT	Hourly Based	€ / Hour	Annual Agreements	€ / Year	Project Based	€ / Project
For Finance - Stock Exchange	-		-		✓	25000.00
For Finance - FX	-		-		✓	19000.00
For Finance - Coin Market	-		-		✓	19000.00
For B2B Solutions	-		-		✓	19000.00
For B2C(Solutions for Enterprenuers)	-		-		✓	19000.00

ACADEMIC RESEARCH AND CONSULTANCY	Hourly Based	€ / Hour	Annual Agreements	€ / Year	Project Based	€ / Project
Academic Researches & White Papers	✓	79.00	-	-	✓	6900.00
AI Strategy Board Consultancy	✓	Free	✓	Free	✓	4900.00
Academy of Emotional Intelligence	✓	49.00	✓	9900.00	✓	4900.00
In-house Bootcamps	✓	299.00	-	-	✓	29000.00