**Week 3**

* Software Engineering: Software engineering is the process of designing, developing, testing, and maintaining software products using systematic and structured approaches. At its core, software engineering is an engineering discipline. It applies engineering principles, methodologies, and techniques to create and maintain high-quality software systems that meet the needs of users and stakeholders.
* Software dev process -
  + Requirements gathering - This involves identifying and documenting the functional and non-functional requirements of the software.
  + Design (specification) - This involves creating a detailed design of the software based on the requirements gathered in the previous phase.
  + Implementation -This involves writing the code for the software based on the design created in the previous phase.
  + Testing (validation) - This involves testing the software to ensure it meets the requirements and is defect-free.
  + Deployment - This involves releasing the software into production and making it available to users.
  + Maintenance (evolution) - This involves updating and maintaining the software to ensure that it continues to meet the changing needs of users and stakeholders.
* Software engineering encompasses various activities, including software design, coding, testing, documentation, and maintenance. It aims to ensure that software products are reliable, efficient, secure, maintainable, and scalable. Software engineering also involves project management, team collaboration, and communication to ensure that software development projects are delivered on time, within budget, and to stakeholder satisfaction.
* software development life cycle’ (SDLC)
  + Waterfall model
  + Incremental model
* Agile software dev: new releases of the system are created and made available to customers every two to three weeks.
  + Customer Involvement - Customers should be closely involved throughout the development process. Their role is to provide and prioritise new system requirements and evaluate the system's iterations.
  + Embrace change - Expect the system requirements to change, and design the system to accommodate these changes.
  + Incremental delivery - The software is developed in increments, with the customer specifying the requirements to be included in each increment.
  + Maintain simplicity - Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
  + People, not process - The skills of the development team should be recognised and utilised. Team members should be left to develop their own ways of working without prescriptive processes.
* Scrum / sprints - A scrum team is a self-organising interdisciplinary team consisting of a product owner, a scrum master, and a small development team (3-6 people). At the project’s inception, the team compiles a product backlog. At the start of each sprint, a subset of the product backlog (sprint backlog) is selected, and the team aims to finish it in a sprint. The process is punctuated by sprint planning and review meetings, and the team’s progress is calculated as velocity and monitored by a ‘scrum master’ and product owner.
  + Development team - A self-organising group of software developers should be no more than seven people. They are responsible for developing the software and other essential project documents. The development team can include software testers.
  + Potentially shippable product increment - The software increment that is delivered from a sprint. The idea is that this increment should be ‘potentially shippable’, meaning that it is in a finished state and requires no further work, such as testing, to incorporate it into the final product. In practice, this is not always achievable.
  + Product Backlog - This is a list of ‘to-do’ items that the scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are required, such as architecture definition or user documentation user stories, or descriptions of supplementary tasks that are required, such as architecture definition or user documentation.
  + Product owner - An individual (or possibly a small group) whose job is to identify product features or requirements, prioritise these for development, and continuously review the product backlog to ensure that the project continues to meet critical business needs. The product owner can be a customer but might also be a product manager in a software company or other stakeholder representative.
  + Scrum - A daily meeting of the scrum team that reviews progress and prioritises work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team. Whilst blockers are mentioned in the scrum, any solutions to blockers must be discussed outside of the scrum to ensure that the scrum is short and efficient.
  + Scrum Master - The scrum master is responsible for ensuring that the scrum process is followed and guides the team in effectively using scrum. He or she is responsible for interfacing with the rest of the company and ensuring that the scrum team is not diverted by outside interference. The Scrum Master is also responsible for chairing each of the meetings involved with the Scrum process. The scrum developers are adamant that the scrum master should not be considered a project manager. Others, however, may not always find it easy to see the difference.
  + Sprint - A development iteration. Sprints are usually 2 to 4 weeks long.
  + Velocity - An estimate of how much product backlog effort a team can cover in a single sprint. Understanding a team’s velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.
* DevOps (development and operations) is an agile development philosophy, practice, and set of tools that enables seamless integration of software development processes and software testing/deployment operations. DevOps has become an integral part of modern software development, where continuous integration and delivery have become the central tenets.
  + 1.**Plan**: in this stage, the development team defines the objectives, requirements, and scope of the software project. They create a roadmap or backlog of tasks and prioritise them based on business value and urgency.
  + 2. **Create:** in this stage, the development team creates the software code using best practices and coding standards. They use version control systems to manage code changes and write automated tests to ensure code quality.
  + 3. **Verify:** in this stage, the software is tested to ensure it meets the defined requirements and standards. Automated tests are executed to detect and fix defects early in the development cycle.
  + 4. **Package:** in this stage, the software is packaged into deployable artefacts or containers using build automation tools. These artefacts contain all the necessary components and dependencies for the software to run in production.
  + 5. **Release:** in this stage, the software is deployed to the production environment using automated deployment tools. Configuration management tools automate the infrastructure changes required for the deployment.
  + 6. **Configure:** in this stage, the software is configured to run in the production environment using configuration management tools. This involves setting up the required hardware, software, and networking components and ensuring that the software is optimised for performance and scalability.
  + 7. **Monitor:** in this stage, the software is monitored continuously to detect and resolve any issues that arise. Monitoring tools collect data on system performance, usage, and behaviour and provide real-time feedback on the software's performance and behaviour.

Devops example:

* An example of DevOps in action could be a software development team using continuous integration and delivery (CI/CD) tools to automate the software delivery process.
* In this example, the development team would use a version control system, such as Git, to manage code changes and collaborate on code development. They would also use a branching strategy, such as Git Flow, to ensure smooth reintegration when merging code. Finally, they would write automated tests for the code using testing frameworks such as XUnit, JUnit or Pytest to ensure the quality of the code.
* The team would then use a CI/CD tool such as Jenkins or Travis CI to automatically build, test, and deploy the code changes to a staging or production environment. The CI/CD tool would also automate the deployment of the code changes to the infrastructure using configuration management tools such as Ansible or Puppet.
* The team would use monitoring tools such as Prometheus or Grafana to monitor the performance and behaviour of the software in production. It would receive feedback on any issues or errors that arise. The team would then use this feedback to improve the software and the DevOps process.

A diagram of software development

Description automatically generated

**AI Engineering**: originating from a data science/data engineering background, this refers to the process of building AI-enabled systems. The AI-enabled systems in this context are typically cloud-based production systems.[[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249251&forceview=1" \l "footnote-1)

**Monitoring**: this refers to monitoring data in data pipelines to ensure that the important attributes such as data quality, latency etc. are collected and actioned upon

**AIOps**: short for AI Operations. This refers to using AI to make automated decisions in operations.

**DataOps**: refers to the use of agile methods and automation in business data analytics.

**MLOps**: refers to the technical infrastructure for operating ML-based products and managing their updates

Al systems challenges –

* The majority of AI-enabled systems fail to take into account the goals of the overall system. AI is typically a small part of the system and an over-reliance or hyper-focus on AI (at the cost of ignoring the over-arching objectives of a system) can result in a system that fails to deliver on its overall objectives. A better approach is to carefully consider system-wide factors, such as the overall objectives, design and maintenance of a system— and then focus on the role of AI and its interactions with other system components.
* In the majority of AI/ML projects, the focus remains on the ML model development and evaluation (e.g. metrics) and not enough attention is given to putting ML models into production or how the ML model lifecycle (for more details see Week 3) relates to the overall software lifecycle.
* When the design of an AI-enabled system is not approached from a systems perspective, it creates unanticipated system-level failures such as data-dependent behaviour, shared resource dependencies, and a misaligned runtime environment for AI component
* Furthermore, AI-enabled systems are more data-centric (e.g., they learn from the data, in contrast to software systems where data is processed only) than non-AI software systems, which require quality issues such as monitorability, security, privacy, explainability, and sustainability. Therefore, any design of the AI-enabled system must address both system and data-related attributes.
* AI-enabled systems degrade at a different rate than the rest of the system components due to issues such as data drift (for a detailed discussion, see Week 7). The best practice is to ensure that the monitoring of the quality of data or performance of ML components is planned for, built-in, and appropriately integrated with AI components.
* AI-enabled systems have different kinds of changes and different rates of changes. For example, a typical AI-enabled system has multiple data sources (or pipelines), and the attributes of each of the dataset (such as size, quality) has a different rate of changes. Changes in one dataset can cause unpredictable change propagation in the system. Furthermore, changes in, and the evolution of, ML models and metrics further complicate such issues. These issues, while identified, are not well understood, and there is no universal or systematic approach to tackling them.
* AI-enabled systems include two systems: the pipeline that produces the ML model and the non-ML system that relies on the ML model. It is important that both systems are developed and remain coordinated. The best recommendations are to (i) co-architect both systems (so that the design is influenced by both ML and non-ML requirements), and to (ii) co-version so that there is a clear understanding and end-to-end traceability of prediction; from training datasets to parameters to model to evaluation dataset to results to deployed model

**Week 4**

'Machine learning life cycle' refers to the process of designing, building, testing, and optimising machine learning models to solve specific problems. It involves a set of techniques and practices that help data scientists and engineers create models. There are nine key activities when it comes to model engineering

A diagram of data engineering

Description automatically generated

* Data collection
* Data cleaning
* Data labelling
* Feature selection
* Feature extraction
* Feature construction
* Model building
* Model training
* Model tuning
* Model eval
* Model deploy
* Model monitoring

**Auto ML**

The goal of AutoML is to automate the repetitive and time-consuming tasks involved in building a machine learning model, allowing data scientists to focus on more complex tasks such as selecting appropriate features or interpreting the results of the model. AutoML uses a combination of machine learning techniques, statistical methods, and optimisation algorithms to automatically search for the best model architecture, hyperparameters, and feature engineering methods.

- Data preprocessing

- Feature engineering

- Model evaluation

- Hyperparam tuning

- Model deployment

**MLOPS**

* **Continuous Integration (CI)**is not only about testing and validating code and components but also testing and validating data, data schemas, and models.
* **Continuous Delivery (CD)** is not only about a single software package or a service but a system (an ML training pipeline) that should automatically deploy another service (model prediction service).
* **Continuous Training (CT)** is a new property, unique to ML systems, that’s concerned with automatically retraining candidate models for testing and serving.
* **Continuous Monitoring (CM)**is not only about catching errors in production systems but also about monitoring production inference data and model performance metrics tied to business outcomes.

**Week 5**

* Charactaristics of cloud computing
  + Broad network access
  + On demand self service
  + Resource pooling and virtualisation
  + Rapid elasticity
  + Measured service
* virtualisation
* Public / private / hybrid /community cloud
* Cloud service models
  + Infrastructure as a service
    - Infa setup
    - Data processing and anal
    - Scalability
    - High avail and disaster recov
    - Management and monitoring
    - **Types of IAAS compute**
      * General perpose compute
      * Compute optimised
      * Memory optimised
      * Storage optimised
  + Platform as a service
    - Eg google app engine
    - Development
    - Deployment
    - Scalability
    - Managed services
    - Moinoting managment
  + Software as a service
    - User centric
    - No programming
  + Contanerisation
  + Serverless computing

**Containerisation** is a lightweight virtualisation method that allows applications and their dependencies to be isolated and packaged into self-contained units called ‘containers’. Each container provides a consistent and reproducible environment for running applications, regardless of the underlying infrastructure. Containerisation is made possible by a container engine or runtime, such as Docker or Kubernetes, which manages the lifecycle of containers. Containers are lightweight because they share the host OS, allowing multiple containers to run on the same host without needing full OS virtualisation (unlike VMs, which need full OS virtualisation). This results in more efficient resource utilisation, faster startup times, and reduced overheads compared to traditional VMs. - **containers as a service’ (CaaS)**

A screenshot of a computer

Description automatically generated

**Serverless computing**

'Serverless computing' is a cloud computing execution model that allows developers to write and execute code without the need to manage or provision servers. In serverless computing, the cloud provider dynamically manages the allocation of computing resources, automatically scaling them up or down based on the demand. Serverless computing revolves around the concept of event-driven execution. Instead of running applications on continuously running servers, serverless applications are triggered by specific events or requests, such as HTTP requests, database updates, or time-based schedules. The application logic is encapsulated in small, stateless functions [[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249280&forceview=1" \l "footnote-1). Having defined containerisation and serverless computing, we now include the two recent innovations in the cloud service model.

**Container as a Service (CaaS) -**CaaS provides a platform for running and managing containers, which package applications and their dependencies into a standardised unit for easy deployment and scalability. Examples include AWS Elastic Container Service (ECS), Google Kubernetes Engine (GKE), and Azure Container Instances (ACI).

**Function as a Service (FaaS) -**Based on serverless computing principles, FaaS allows developers to write and deploy code as functions or serverless applications without managing the underlying infrastructure. Examples of FaaS include AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions.

**A diagram of software components

Description automatically generated**

**Storage**

* Block storage
* File storage
* Object storage

**Hardware for ML**

* GPUs
* Tensor processing units (TPUs)
* FPGAs
* Application specific integrated circuits
* Neural processing units
* Digital signal processing
* Quantun processing units

**PAAS**

* Data services
  + RDBMS
  + Data warehouse
  + Document DB
  + Column DB
  + Graph DB
  + Key value store
* Middleware
  + Messaging queue
  + Streaming
  + API management
  + Integration and workflow
  + ETL processing
* Analytics
  + Big data processing
  + Machine learning framework
  + Visualisation
  + Data discovery

**Application services**

* Search
* Identity and access management (iAM)
* Email discovery
* Notification
* Blockchain
* IOT

**Operational services**

* Deployment
* Devops tools
* Patch management
* Monitoring
* Logging

**Week 6**

Chapter 2 introduces an overview of the challenge of forecasting and the use of AWS Forecast for that purpose. As you read through the chapter, pay attention to:

* How the AI/ML challenges that are common to multiple domains are well-recognised by companies who provide boiler-plate architecture/AI services (e.g., Amazon Forecast in this context, another example is Recommendation Engines) to rapidly build an AI-enabled system around it. It is always best to explore such ‘off-the-shelf’ AI/ML solutions before considering any in-house development of an AI-enabled system.
* How the overall architecture of an AI-enabled system can be built using both ML/non-ML (e.g., monitoring, storage) services available on a platform.
* How services are priced and dependent upon your requirements such as latency etc, which in turn are informed by the specification of your problem (see “Framing your forecasting problem” in Chapter 2 (Hoarau,2022) [[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249292" \l "footnote-1).

Coincidentally, an example of a video-on-demand service is discussed in Chapter 2 which has some parallels to the coursework in Week 8. It will be useful to give some thoughts and consideration to how some discussions (e.g., requirements, architecture, datasets, pricing etc) could be applicable to your coursework

Chapter 3 discusses data preparation, ingestion and analysis and provides a detailed process for data ingestion on AWS.

You should pay attention to:

* How AWS storage services can be used to create ‘data buckets’ (S3) based on your requirements and how its default configuration/policies and roles could help speed up or facilitate the process[[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249293&forceview=1" \l "footnote-1),
* Why it matters that such buckets are stored in certain regions/parts of the world.
* How the web interface provides an easy way to explore your datasets and their properties (e.g., schemas), and modify them without the need for any code etc.
* Besides the web interface, is there another way to access or automate the provision of storage services? (Hint: Amazon CLI)
* How exactly stored data could be connected/ingested by another service (e.g., ML algorithm).

Chapter 4 [[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249294&forceview=1" \l "footnote-1) discusses using ML algorithms for forecasting and how AutoML could be used.

You should pay attention to:

* How multiple predictors could be easily trained once the dataset is connected.
* How common attributes/parameters (e.g., forecasting horizon or optimisation metrics) associated with predictors are readily identified by the platform and made available for you to set 'on the fly' as per your requirements.
* How AutoML eliminates the need for specifying predictors/its parameters.
* How AutoML can provide a comparative analysis (e.g., via a Dashboard in the example) to help choose the best-performing predictor.
* How measures such as explainability could also be explored on such platforms.

For this lesson, you will need to read through Chapter 1 and Chapter 2 in Potgieter (2022) [[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249295&forceview=1" \l "footnote-1). We note that their approach is more code-centric when compared with Hoarau (2022) [[2]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249295&forceview=1" \l "footnote-2).

Chapter 1 outlines an overview of the ML process and provides a step-by-step process to turn a business problem or goal into an ML challenge.

You should pay attention to:

* What activities are carried out at each stage of a typical ML process (Figure 4).
* How their ML process workflow (Figure 4) compare with the ML lifecycle we discussed in Week 4 (Figure 1, Week 4).
* How a model can be tuned by determining the best parameters to tune (note, in general, this is an advanced topic)
* How the optimised model could be deployed into production and how platform services (e.g., CodeCommit) can help streamline this process.

Chapter 2 builds on Chapter 1 to discuss the broader landscape of AI services and its layers (Figure 5), followed by concrete implementation of the system under question. 

You should pay attention to:

* How platforms offer AI services to address common ML challenges (e.g., computer vision), and how each layer is dependent on underlying layers.
* How AI-enabled systems can make use of existing AI services to acquire AI functionality without the need to develop or have a dedicated ML component as part of their system.
* How products such as SageMaker Studio can bring the entire ML process (i.e. data ingestion or training to the prediction or insights) into one place, further increasing automation and ease of developing AI-enabled systems.
* How AutoML workflow in Week 4 (Figure 3, Week 4) maps onto the AutoML process used by SageMaker Autopilot (outlined in Figure 2.18 in Chapter 2 Dahlberg, 2022 [[1]](https://engage.bath.ac.uk/learn/mod/page/view.php?id=249296&forceview=1" \l "footnote-1))

**Week 7**

**Software architecture** is the blueprint for building software systems. It defines how the system is structured, the components involved, how they interact, and their behavior. According to the IEEE, it is “the fundamental organization of a software system, including its components, their relationships, and the principles guiding its design and evolution.”

Architecture is important because it strongly influences the **non-functional properties** of a system. These are also known as **quality attributes**, and they describe how well a system performs, rather than what it does.

Here are some common non-functional properties for online systems:

**Attribute** **Key Question**

**Responsiveness** Does the system respond to users quickly?

**Reliability** Does it work as expected for both developers and users?

**Availability** Is it available whenever users need it?

**Security** Is it protected from unauthorized access and attacks?

**Usability** Is it easy for users to access and use the features?

**Maintainability** Can it be updated or improved easily and affordably?

**Resilience** Can it keep working even if parts fail or it’s under attack?

**Key principles of software architecture design**

* **Modularity**
* **Abstraction**
* **Seperation of concerns**
* **Encapsulation**
* **Flexability and Extensibility**
* **Scalability**
* **Loose Coupling**
* **High Cohesion**

**Arcitecture examples**

https://en.wikipedia.org/wiki/Easyrec

A diagram of a software system

Description automatically generated

**MLOps Architecture Guide**

<https://neptune.ai/blog/mlops-architecture-guide>

1. Common design patterns for ML-based software architecture, their trade-offs and guidance around choosing an appropriate architecture.
2. How requirements (functional and non-functional) are established from a project description (Section: Best MLOps architectures on a defined project).
3. The structure of architectural diagram (section: Defining system structure)
4. The potential considerations are important at the implementation, deployment, and monitoring stages.

**Data drift**

Machine learning (ML) models can degrade over time due to changes in data, known as **data drift** and **concept drift**.

• **Data drift** occurs when the data used in production changes from the data the model was trained on, making the model’s predictions less accurate.

• To handle data drift, it’s crucial to continuously update training data, retrain models, and adjust how the model weights different inputs.

• An example of data drift is in weather prediction, where changes in reporting methods or technology (like weather radars) can lead to discrepancies between older and newer datasets. Climate change is another factor that influences how well a model can predict current conditions based on outdated data.

• Drift can happen for several reasons, such as changes in data schemas, feature distributions, or the meaning of the data. Changes like business updates or new bugs from software updates can also impact model performance.

• **Continued model evaluation** is necessary to detect and address these issues before the model becomes unreliable.

In short, data drift requires regular monitoring and retraining of models to ensure they stay accurate over time.

**Concept drift**

**Concept drift** happens when the relationship between ML model inputs and the target changes, often due to evolving adversarial behaviors, such as in fraud detection, spam filters, and cybersecurity.

• For example, credit card fraud detection models need to adapt as fraud tactics shift, like when chip and pin technology led to more online fraud. Without updates, models will perform poorly as fraud patterns change.

To address **concept drift** (or **data drift**), the model must be retrained with updated data. However, the challenge is knowing **when** and **how often** to retrain, since these processes are time-consuming and costly.

The best way to detect model deterioration is by continuously monitoring its predictive performance using the same metrics from the development phase. Continuous evaluation helps ensure that the model and any updates perform as expected.

**Developing telemetry**

* **Metric -** Defining the quality metric of interest; e.g., prediction accuracy, recall, daily revenue, etc.
* **Data collection -** Identifying the data that can be collected in production; e.g., user activity from log files
* **Operationalisation -** Describing how the objective metric is derived from the collected data
* In many cases, direct measurement of desired metrics isn’t possible, so **proxy metrics** are used instead. For example, in transcription services, instead of traditional NLP accuracy metrics, proxy metrics like **average star ratings** or **percentage of corrected words** can be applied. These metrics are collected via **telemetry data**, and techniques like sliding windows help compute averages over specific periods.
* **Choosing the right data to collect** is challenging due to the large volume of production data. Not all data needs to be collected, such as in video conferencing where only some extracted features are necessary, not full video files. **Sampling** can reduce data collection but may be difficult if events are rare.
* The **system architecture** affects telemetry design. Cloud systems can collect data centrally, while other systems like mobile apps or IoT devices may face challenges due to connectivity or privacy concerns. Privacy policies also play a major role, such as Amazon’s approach with Alexa versus Apple’s stricter data policies.
* Finally, designing a **user interface (UI)** that collects data without disrupting the user experience, and managing data collection when systems are offline, are important challenges. Overall, telemetry design is complex and has a significant impact on system quality.

Once the telemetry system is in place, monitoring and experimentation help assess model quality and system success. The first step is to set up a **monitoring system** that tracks performance over time and across different groups. This helps detect slow degradation, such as changes in user behavior. Retraining the model with new data (often from telemetry) can fix this issue.

Tools like **Grafana** and other log data collectors (e.g., **Prometheus, Logstash**) can be used to monitor performance, and system alerts can notify when metrics exceed certain thresholds.

For testing, **A/B experiments** are a common approach. Users are divided into groups to test different product versions, like testing two machine learning models for film suggestions. These experiments are easy to run with large user bases, as internet-connected services provide many participants. This makes it easier to get reliable results without concerns about unbalanced groups.

**Feature flags** are used to control which users see the new version or the old version. They allow developers to implement two variants in one system, using simple code to switch between them based on the user group.

When running **A/B experiments**, users are typically **randomly assigned** to groups to avoid bias, and their group assignment should remain stable throughout the experiment. Groups can be formed simply or in more complex ways, such as dividing users into categories (e.g., beta testers, developers)

It’s essential to **collect telemetry data** for each group separately to compare results (e.g., sales from users who saw predictions from a new model vs. the old one). After the experiment, statistical tests like **confidence intervals** and **Student’s t-tests** are used to analyze if the new version caused differences in performance.

A/B experiments typically involve **large groups** of participants and run over several hours or days to ensure **statistical significance**. However, a balance is needed between having enough users to ensure accurate results and minimizing exposure in case the new model performs poorly. In practice, companies often run experiments with small percentages of users (0.5%-10%) until significant results are observed. Large companies can run many experiments simultaneously with minimal user exposure.

**Canary releases** and **A/B experiments** are similar technically, but have different goals. A/B experiments evaluate changes in product or model performance, while canary releases are used to gradually and safely roll out new changes to more users based on telemetry data. Both use techniques like feature flags and load balancing, and their infrastructures can be integrated.

Other types of experiments in production include:

• **Shadow releases**: Two versions run in parallel, but only the old version’s predictions are shown to users while the new version is tested for stability.

• **Blue/green deployments**: Both versions are deployed, but traffic switches entirely to the new version, with the option to quickly roll back if needed.

• **Chaos engineering**: Tests system robustness by simulating failures or issues like latency to ensure the system can handle disruptions.

In summary, production systems provide essential test data, and good telemetry and monitoring are key to evaluating model performance. Using tools like statistics and building solid infrastructure ensures the success of various testing methods, including A/B tests, canary releases, and chaos engineering.

**I apologize** for the overly brief summary. You're right to ask for more detail. I'll provide a more comprehensive overview of the aiSTROM framework, while still condensing the full paper:

1. Introduction and Context:

- AI is becoming increasingly integrated into daily life and business processes.

- A survey by Rackspace Technology found that 34% of AI research and development projects fail or are abandoned.

- 31% of companies list a poorly conceived strategy as a reason for failure.

- The paper presents aiSTROM, a Strategic ROadMap framework to guide organizations in developing successful AI strategies.

2. aiSTROM Framework in Detail:

a) Identify Opportunities:

- Create a list of possible goals and problem statements that can be tackled by AI.

- This should be done by a team with both domain/organization knowledge and AI expertise.

- Consider internal efficiency gaps, new AI technologies, competitor strategies, and potential new business models.

- Select the top n projects (typically 3-5) to focus on, maximizing potential impact given the lowest effort.

b) Data Strategy:

- Data is crucial for any machine learning algorithm.

- Consider data sources: Does the organization already have necessary data? If not, start collecting.

- Big data considerations: Volume, Velocity, Variety, Value, and Veracity.

- Data collection vs. acquisition: Weigh the costs and benefits of collecting data vs. purchasing existing datasets.

- Legal issues: Consider privacy laws (e.g., GDPR, CCPA) and security measures to protect data.

- Data storage: Decide between internal storage or external data centers. Consider data lakes vs. data warehouses.

c) AI Team:

- Address the scarcity of AI talent and high demand for these skills.

- Required skills include mathematics, statistics, data mining, big data engineering, pattern extraction, visualization, software engineering, communication, and continuous learning.

- Strategies for obtaining AI talent:

- Upskilling existing employees

- Targeting recent AI graduates

- Collaborating with academic institutions

- Acquihiring (acquiring companies for their talent)

- Focus on retaining employees through continuous education and benefits.

d) Organizing AI Development:

- Decide on team positioning: centralized, decentralized, or hybrid "hub-and-spoke" approach.

- Consider a portfolio approach to spread risk across different projects and timeframes.

- Evaluate AI as a Service (AIaaS) options for non-core technologies.

- Assess whether to develop in-house, outsource, or acquire AI technology.

- Implement agile development methodologies and consider MLOps practices.

e) Technologies:

- Balance accuracy vs. explainability in AI models (XAI considerations).

- Consider human-in-the-loop approaches for scarce labeled data scenarios.

- Decide whether AI should augment or replace human tasks.

- Evaluate cloud vs. in-house hosting options, considering factors like cost, scalability, and control.

f) Key Performance Indicators (KPIs):

- Define clear, value-based metrics to measure AI project success.

- Consider both financial outcomes and value creation for customers.

- Popular KPIs include profit margins, revenue growth, customer satisfaction scores, and process improvements.

- Also consider AI-specific performance metrics like accuracy, confusion matrices, or F1-scores.

g) Risk Assessment:

- Consider risks such as:

- The stochastic nature of AI algorithms

- Biases and ethical concerns in AI models

- Security vulnerabilities, including adversarial attacks

- Strategic decisions around team structure and data management

- Perform a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis for each potential project.

- Consider implementing a portfolio of projects with varying risk levels.

h) Enabling a Cultural Shift:

- Educate employees across the organization about AI technologies and their potential impact.

- Consider setting up an AI center of excellence to create awareness and knowledge.

- Promote a culture of adoption and continuous learning to support widespread AI integration.

3. Conclusion:

The aiSTROM framework provides a comprehensive approach for organizations to develop and implement successful AI strategies. It guides managers and developers through various challenges in the implementation process, from identifying opportunities to enabling a cultural shift. By thoroughly analyzing potential AI projects using this framework, organizations can make informed decisions that are likely to increase the success of their AI initiatives.

This more detailed summary provides a comprehensive overview of the aiSTROM framework while still condensing the full paper. It captures the main structure and key points of each section in the framework.

Certainly. Here's a detailed summary of the article "How to Define and Execute Your Data and AI Strategy" by Ulla Kruhse-Lehtonen and Dirk Hofmann:

1. Introduction:

- Many organizations recognize that future success depends on data and AI capabilities.

- Companies are investing heavily in this area, but many have become disillusioned in their journey to create companywide, data-driven business transformation.

- The article discusses common pitfalls and provides recommendations for business leaders.

2. Setting the Data and AI Vision:

- Data and AI priorities should be derived from business priorities.

- Consider the business case for each area when assessing where to focus data and AI efforts.

- Use a data opportunity matrix to guide the process, starting with optimizing current business processes.

- Consider new data-driven business opportunities, including data as a business and data partnerships.

3. Data Management and Data Governance:

- High-quality data is the foundation for successful, productized AI.

- Data should be structured according to FAIR principles (Findable-Accessible-Interoperable-Reusable).

- Start building the data asset with data needed for prioritized business opportunities/use cases.

- Conduct a data due diligence or data inventory to assess the current state of data assets.

4. Solution Architecture and Technology:

- After defining the business & AI vision and conducting data due diligence, define the target architecture and its development roadmap.

- Consider end-to-end use case logic, including data collection from operating systems, data warehouses, cloud environments, and business-interfacing systems.

- Managing the transition from traditional IT systems to the digital world can be lengthy and may initially increase costs.

5. Data and AI Protection, Privacy, and Regulation:

- Comply with data protection regulations like GDPR.

- Write privacy policies according to the desired state of AI use cases, not just current ones.

- Consider transparency and explainability of AI solutions, especially in light of potential new regulations.

6. Human Skills:

- New roles are needed at four levels: business units, data science, data management, and data platforms/technical solutions.

- The AI strategist role is critical for translating business vision into data and AI requirements.

- Hire a balanced data science team with diverse backgrounds.

- Don't neglect technical roles like data engineers and data architects.

- Use assessment tests in recruiting to distinguish candidates' skills.

7. Data and AI Organization:

- The optimal structure depends on company size, culture, AI maturity, and types of data/AI tasks.

- Establishing a center of excellence (CoE) can help bring focus to the topic.

- In mature companies, the CoE's role may diminish as data use becomes widespread.

- Consider introducing a temporary companywide AI program to drive the agenda forward.

8. Operating Model:

- Establish an AI steering group or include data and AI development in existing leadership team meetings.

- Consider centralizing budgets for the first years of the CoE to drive prioritization.

- Align technical and business development processes.

- Give the same incentives to everyone involved in a data/AI project to increase business impact.

9. Data Science and Machine Learning/AI Algorithms:

- Treat algorithms as an asset, building a FAIR portfolio over time.

- Establish maintenance processes for data and algorithm assets.

- Address the explainability of machine learning algorithms, especially for "high-risk" applications.

10. Execution Steps:

- Translate business strategy into data and AI vision.

- Identify business processes for data and AI use.

- Understand current state of data and AI capabilities.

- Describe target state for business processes with data and AI capabilities.

- Define new data-driven business and product ideas.

- Define execution roadmap and investments.

- Execute first use cases by creating an AI playbook.

- Automate and scale up operations.

The article concludes by emphasizing that the highest level of AI maturity is when the whole company moves in unison, using data and AI as part of daily business, with humans ensuring automation is d**one smartly.**