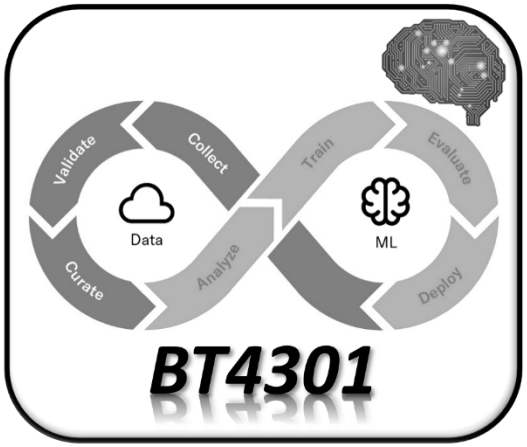



# Lecture 4

# Overview of DevOps, DataOps and MLOps

BT4301 – Business Analytics Solutions Development and Deployment
AY 2023/24 Semester 2

**Lecturer**: A/P TAN Wee Kek

**Email**: tanwk@comp.nus.edu.sg :: **Tel**: 6516 6731 :: **Office**: COM3-02-35

**Consultation**: Wednesday, 9:30 pm to 10:30 pm. Additional consultations by appointment are welcome.

# Learning Objectives

- At the end of this lecture, you should understand:
  - Overview of software engineering process.
  - DevOps
  - Implementation and toolchain for DevOps
  - Overview of analytics process.
  - DataOps
  - MLOps

# Readings

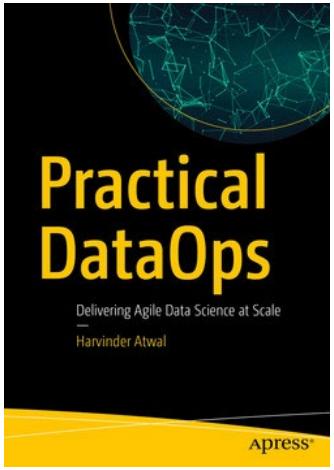
- Reference Book 4:

  **Practical DataOps – Delivering Agile Data Science at Scale**

  Authors: Harvinder Atwal

  1st Edition (2019), Apress

  https://linc.nus.edu.sg/record=b4034242



# Readings (cont.)

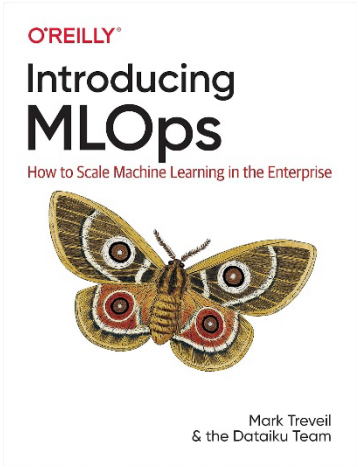
- Reference Book 5:

  **Introducing MLOps**

  Authors: Mark Treveil and The Dataiku Team

  1st Edition (2020), O'Reilly Media

  NLB Link

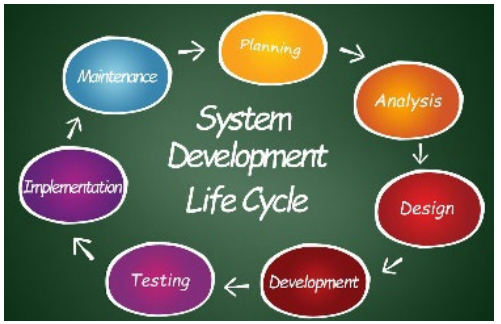


# Readings (cont.)

- Required readings:
  - Nil
- Suggested readings:
  - Reference Book 4 – Chapter 1 and 2
  - Reference Book 5 – Chapter 1

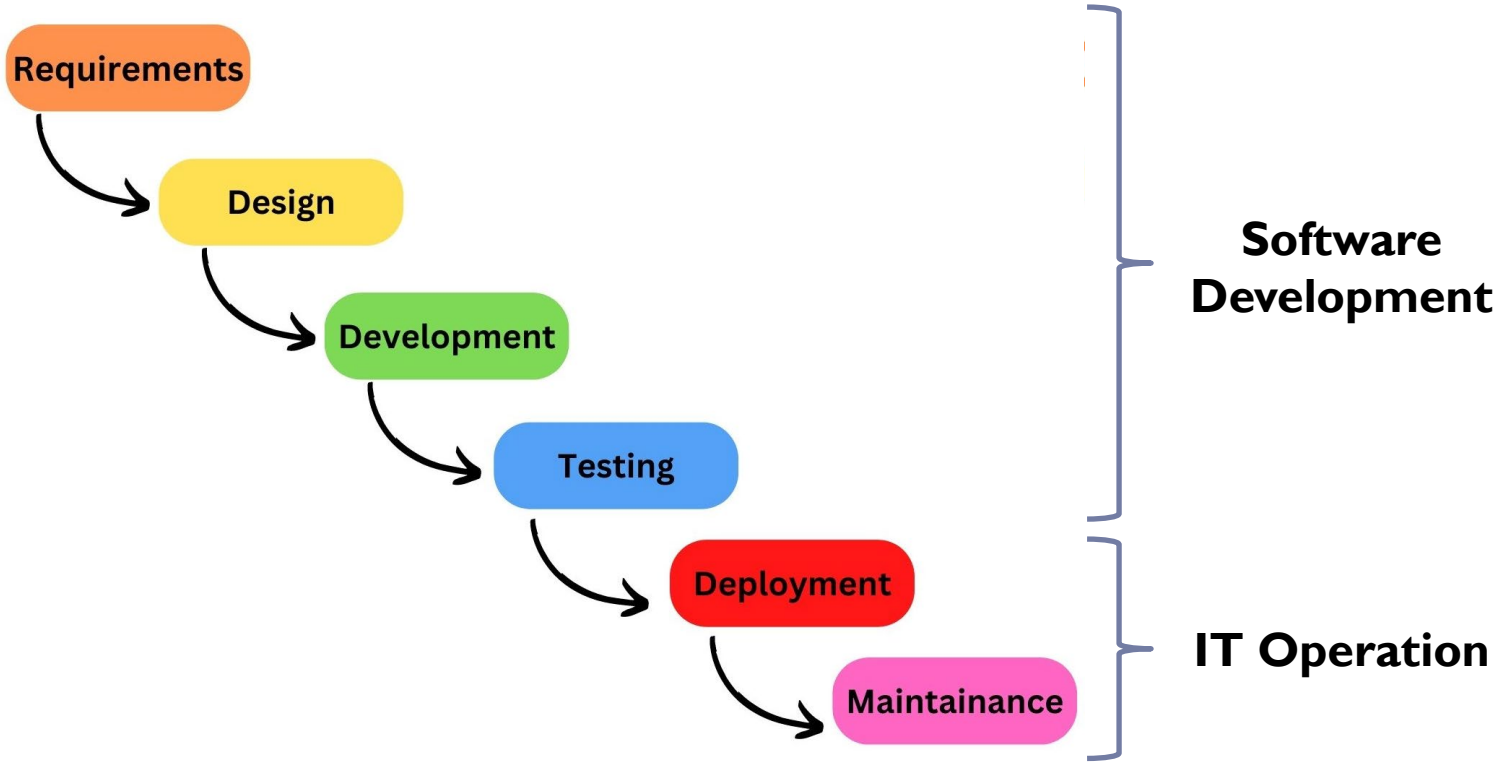
# Software Engineering

- What is **software engineering**?
  - Refers to the process of developing and maintaining large and high-quality software system within constraints.
  - Typical constraints include time, manpower and budget.
- Emphasizes the adoption of engineering techniques in the software development process:
  - Techniques include definition, implementation, assessment, measurement, management, change and improvement of the software life cycle process itself.
  - **Software configuration management** features prominently.



# Software Engineering? (cont.)

- Characteristics of software engineering:
  - Software life cycle process essentially resembles the waterfall model:

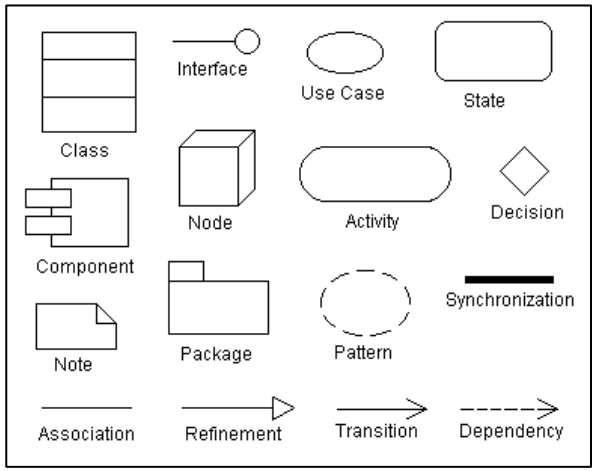


# Software Engineering? (cont.)

- Software testing is done towards the end of the main development process.
- Clear segregation between software development and IT operation.
- Often, the development team and maintenance team are different and lack coordination.

- Requirement analysis and design stages emphasize the adoption of formal documentation:
  - E.g., Unified Modelling Language (UML).

| Artifact | Usage |
|---|---|
| Use Case Diagram | Analysis of usage requirement. |
| Usage Scenario | Exploration of system usage. |
| Activity Diagram | Analysis or design of business process. |
| Class Diagram | Conceptual and logical data modeling and depicting other functional classes (e.g., components or session beans). |
| Sequence Diagram | Modeling the logic of a use case. |

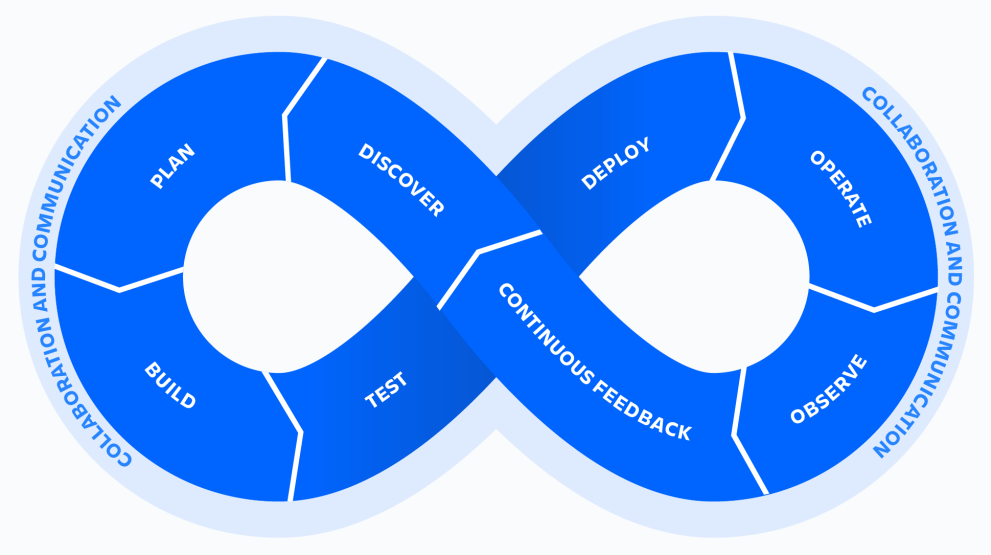


# Software Engineering? (cont.)

- Problems with traditional software engineering:
  - Project centric approach with a finite duration.
  - Development team hands over to operation/maintenance team:
    - 20% to 40% of development efforts is spent on initial development.
    - 60% to 80% of development efforts is spent on subsequent maintenance.
  - What could go wrong when developers who wrote code worked apart from operations who deployed and supported the code?

# DevOps

- What is **DevOps**?
  - A methodology consisting of a set of cultural philosophies, practices and tools.
  - Integrates and automates software development (**Dev**) and IT Operations (**Ops**).
  - Improve software development life cycle.



# DevOps? (cont.)

- Objectives of DevOps:
  - Increases an organization's ability to deliver applications and services.
  - Evolves and improves products at a faster pace.
  - Better serve customers and compete more effectively in the market.

# DevOps Cultural Philosophies

- Adoption of DevOps requires a change in **culture and mindset**.
- DevOps emphasizes:
  - Removing barriers between the development team and operations team.
  - In certain organisations, there is no distinction between development and operations, i.e., engineers take on both roles.
- Development and operation teams work together closely:
  - Optimise the productivity of developers and the reliability of operations.
  - Communicate frequently, increase efficiencies, and improve the quality of services provide to customers.

similar to Agile.

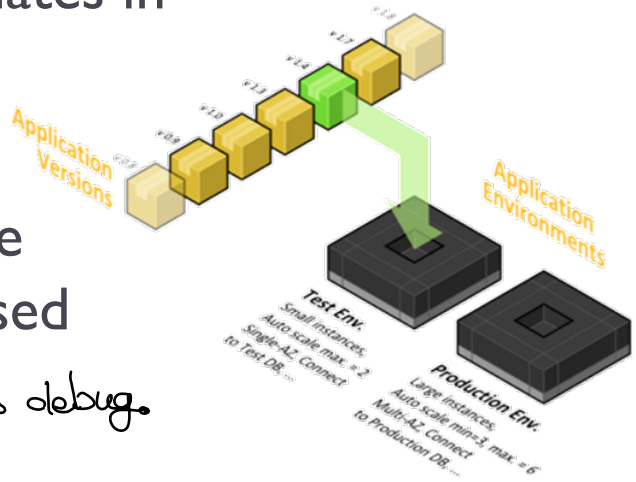
# DevOps Cultural Philosophies (cont.)

- Take full ownership for their services:
  - Think about the needs of end customer.
  - How can both teams contribute to solving those needs.
- Beyond development and operations:
  - Quality assurance and security teams may also be involved and become tightly integrated.
  - In organisations adoption DevOps, all teams view the entire development and infrastructure life cycle as part of their responsibilities.

# DevOps Practices

- **DevOps practices**:
  - Help organisations innovate faster through automating and streamlining the software development and infrastructure management processes.
  - Accomplished with proper tooling.
- **Perform very frequent but small updates**:
  - Incremental updates versus occasional updates in traditional software development.
  - Reduce the risk of each deployment cycle.
  - Help teams address bugs faster due to ease of identifying the last deployment that caused the error.

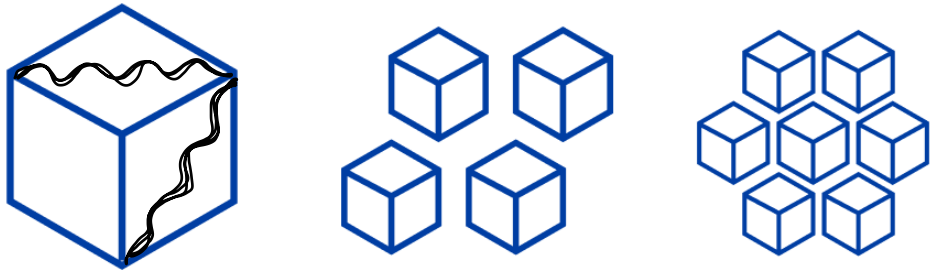
If you only change small amounts of code, it is easier to debug.



# DevOps Practices (cont.)

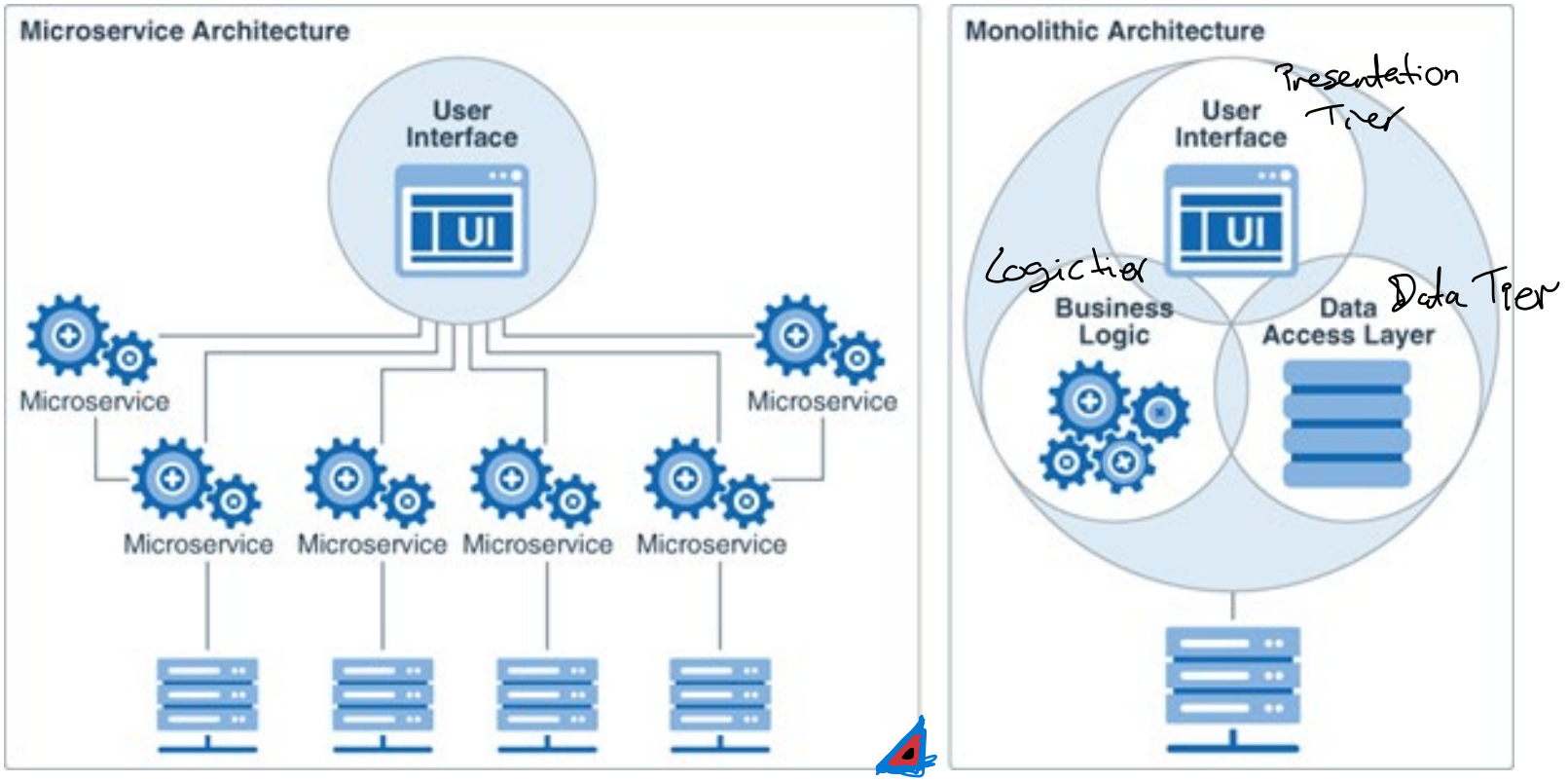
- **Microservices architecture**:
  - Improves the flexibility of applications and enable quicker innovation.
  - Decouples large, complex systems into simple, independent projects:
    - Applications are broken into many individual components (services).
    - Each service is scoped to a single purpose or function.
    - A service operates independently of its peer services and the application as a whole.

Microservices are decoupled
So they can operate
separately.
There is no such thing as
decoupling.



# DevOps Practices (cont.)

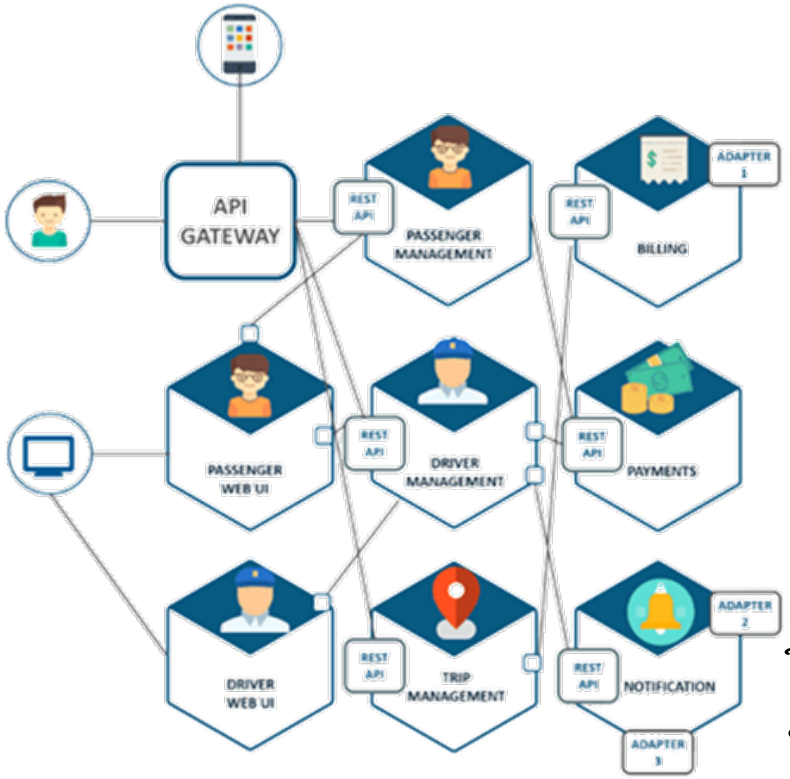
- Reduces the coordination overhead of updating applications:
  - Each service is paired with a small, agile team.
  - The team takes ownership of each service.



# DevOps Practices (cont.)

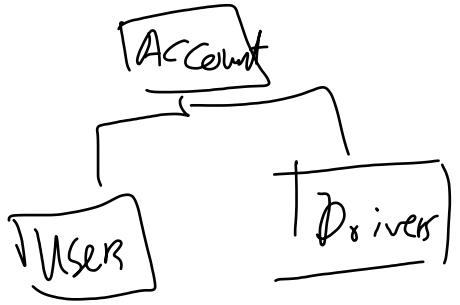
- Example – High-level overview of Uber's microservices architecture:

there is no discoordination of data, because each entity class is inside only one microservice.




What if a person is a driver and a user at the same time?

You can create a class:




Is it that every micro-service only handles an entity class?

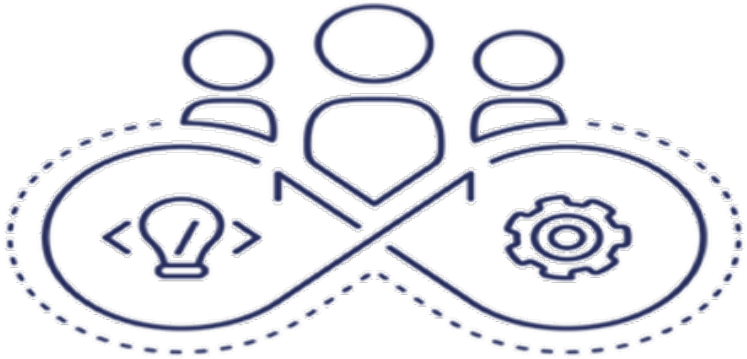
- What are the implications of microservices architecture on analytics?

the entity classes are very close to each other in a micro-service.

But can be several.

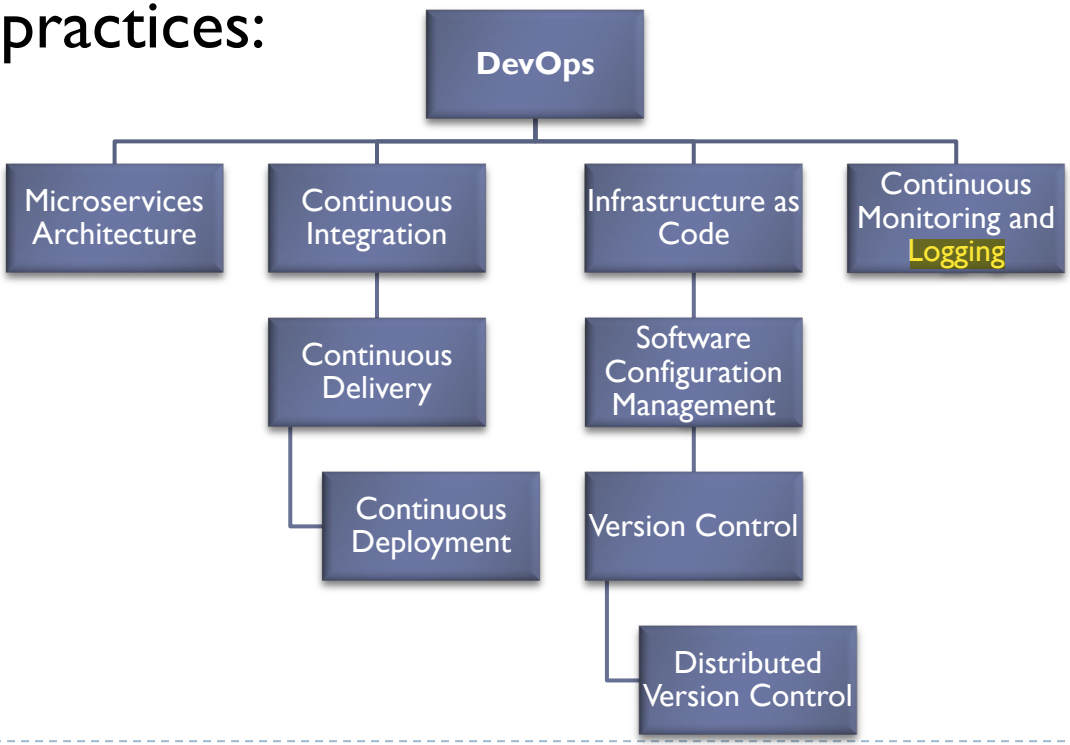
# DevOps Practices (cont.)

- Combination of increased release frequency and microservices architecture presents its own challenges:
  - Leads to significantly more deployments.
  - Requires other complementary DevOps practices:
    - E.g., continuous integration and continuous delivery.
    - Empower organisations to deliver rapidly in a safe and reliable manner.
  - Includes infrastructure automation practices:
    - E.g., infrastructure as code and configuration management.
    - Keeps computing resources elastic and responsive to frequent changes.



# DevOps Practices (cont.)

- Necessitates measurement practices:
  - E.g., monitoring and logging to track performance of applications and infrastructure.
  - React quickly to problems.
- DevOps practices:



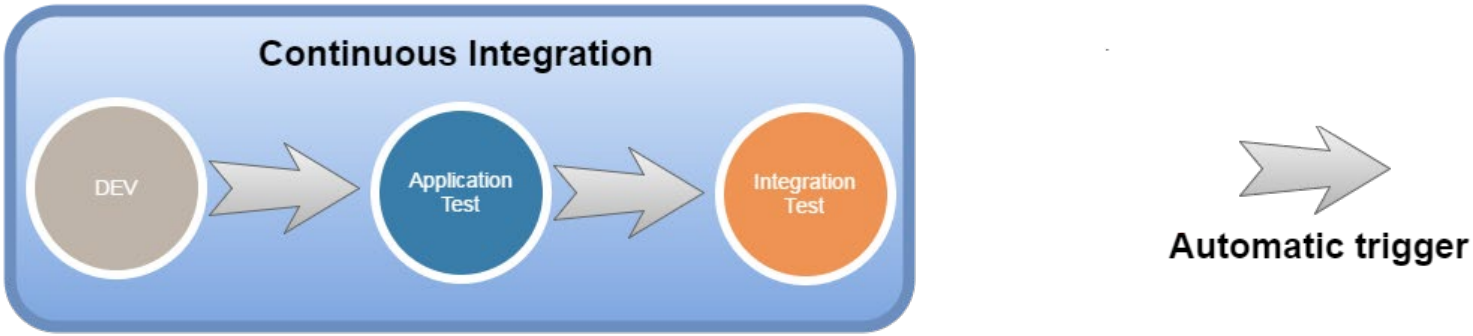
# Continuous Integration

- What is **Continuous Integration (CI)**?
  - Developers regularly merge their code changes into a central repository.
  - The changes are validated by creating an automated build and running automated tests against the build.
- Objectives of CI:

  - Reduce the time it takes to validate and release new software updates.
  - Find and address bugs quicker thus improving software quality:
    - In the past, developers work in isolation for extended period of time.
    - Changes are merged after work is completed.
    - Complicates code merging and bugs tend to accumulate and snowball.

# Continuous Integration (cont.)

- How does CI work? First you commit, and then you push.
  - Developers frequently commit to a shared repository using a version control system such as Git.
  - Prior to each commit, developers may run local unit tests as an extra verification layer before integrating.
  - A continuous integration service automatically builds and runs unit tests on the new code changes.



CD is an extension of CI.

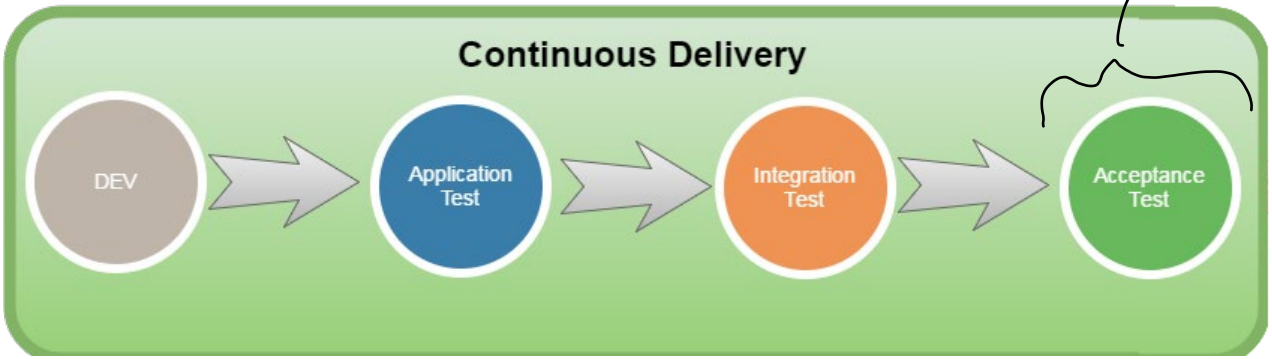
# Continuous Delivery

- What is **Continuous Delivery (CD)**?
  - Code changes are automatically built, tested, and prepared for a release to production. → same to CI
  - Extension of CI by deploying all code changes to a testing environment and/or a production environment after the build stage.
  - If CD is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.
- Objectives of CD:
  - Automate testing beyond just unit tests.
  - Verify application updates across multiple dimensions before deploying to customers

# Continuous Delivery (cont.)

- E.g., integration testing, UI testing, load testing, API reliability testing.
  - si la página aguanta.
  - You test it in Postman.
- Thoroughly validate updates and pre-emptively discover issues.

- How does CD work?
  - Extension of CI.



Now CD normally refers to continuous Deployment.

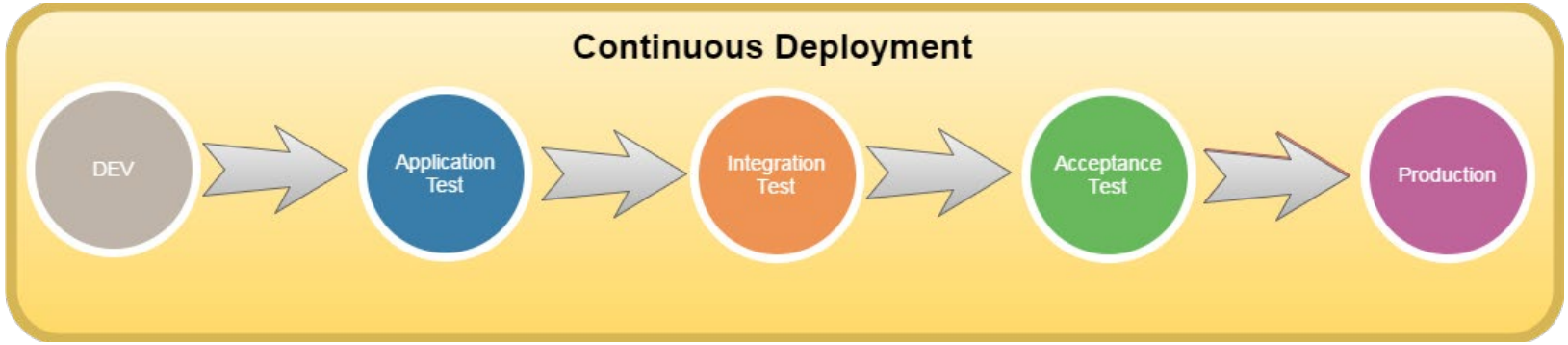
# Continuous Deployment

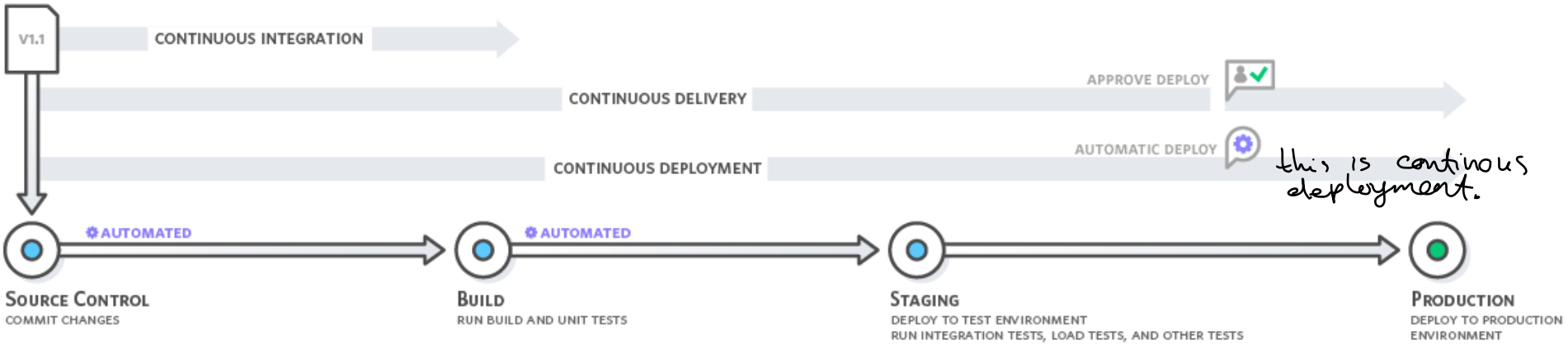
- What is **Continuous Deployment**?
  - Extension of CD.
  - With CD, every code change is built, tested, and then pushed to a non-production testing or staging environment.
  - There can be multiple, parallel test stages before a production deployment.
- Difference between Continuous Delivery and Continuous Deployment:
  - In CD, there is a manual approval to update to production.
  - With Continuous Deployment, production happens automatically without explicit approval, i.e., no human intervention is required.

It carries higher risk, because no human looks at it.

# Continuous Deployment (cont.)

- How does Continuous Deployment work?
  - In general, only a failed test will prevent a new change from being deployed to production.



# Continuous Deployment (cont.)

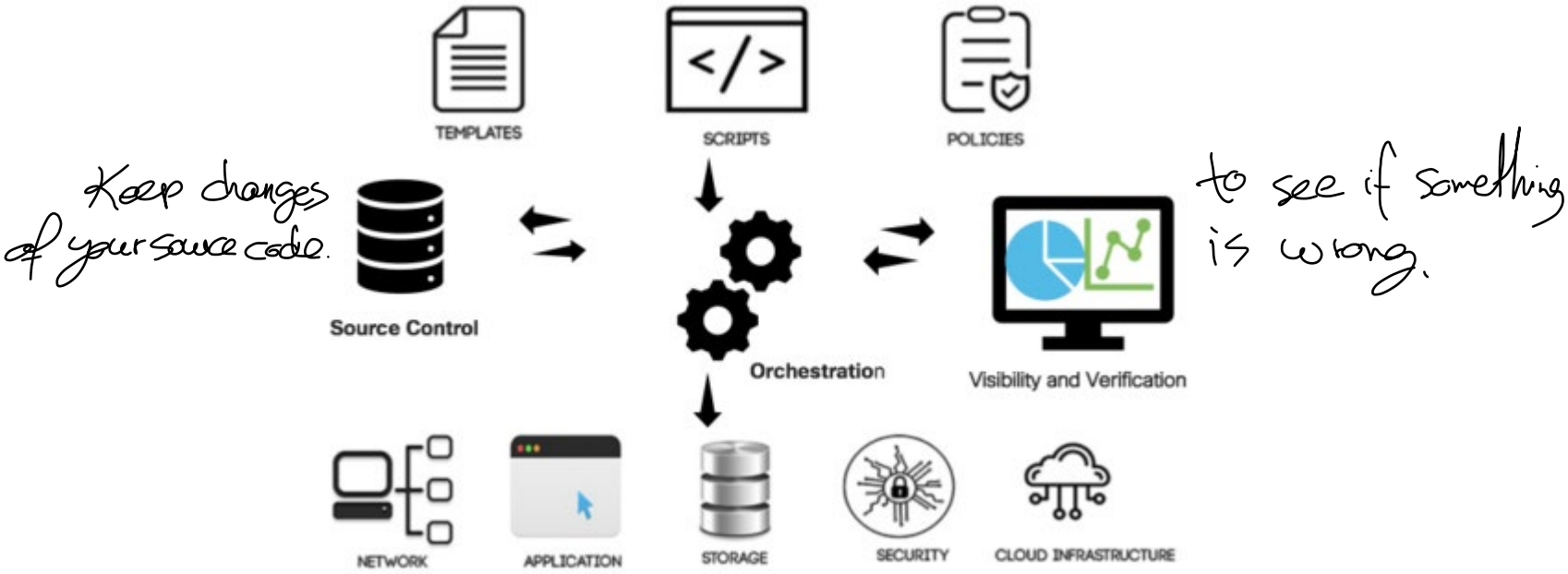
- What are the implications of CI/CD on analytics?
  - Is CI/CD applicable to analytics?

# Infrastructure as Code

- What is **Infrastructure as Code**?
  - Infrastructure is provisioned and managed using code and software development techniques.
  - E.g., version control and CI.
  - APIs from cloud computing service provider enables developers and administrators to interact with infrastructure programmatically, and at scale.
- Objectives of Infrastructure as Code:
  - Negate the need to set up and configure resources manually.
  - Treat infrastructure in a manner similar to application code.

# Infrastructure as Code (cont.)

- Infrastructure and servers can be quickly deployed using standardized patterns and maintained.
- How does Infrastructure as Code work?
  - Closely coupled to the infrastructure.



# Software Configuration Management

- What is **Software configuration management (SCM)**?
  - A key element of Infrastructure as Code.
  - The task of tracking and controlling changes in the software:
  - SCM is part of the bigger software engineering process.
  - It is also part of a larger cross-disciplinary field of configuration management.
- A key components of SCM is **version control**:
  - Version control entails the management of changes to source code.
  - Modern software development typically involves a team of developers.

# Software Configuration Management (cont.)

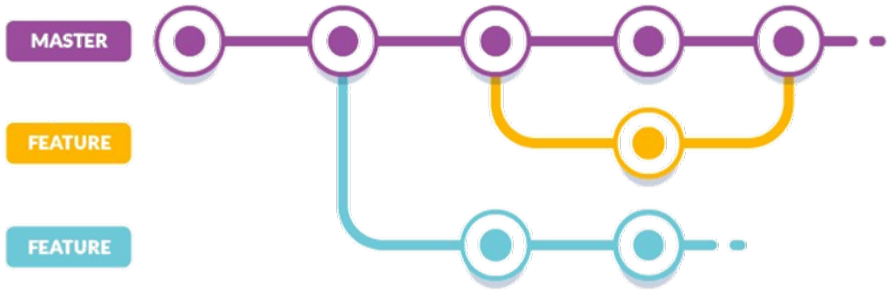
- Different members create and change different files that require integration thus necessitating version control.
- Different members may also change the same files.

- Other important components of SCM:
  - **Build management** – Managing the process and tools used for building the software from the source code:
    - Maven is a build automation tool for Java.
    - A Maven repository holds build artifacts and dependencies of varying types (e.g., library JARs).


  - **Environment management** – Managing the software and hardware that host the software.

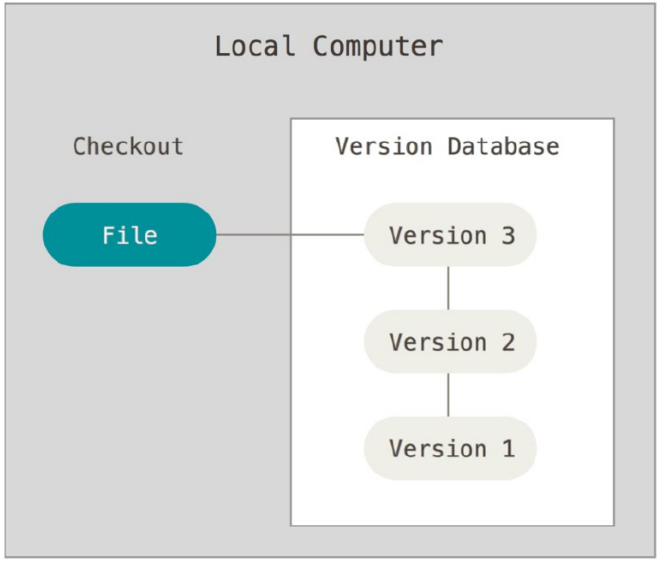
# Version Control

- What is **Version control**?
  - Systematic approach of recording changes to a file or set of files over time.
  - Recall specific versions of the files at a later juncture.
  - Can be applied to nearly any type of file.
  - Primarily interested in software source code files.
- **Version Control System (VCS)** enables you to:
  - Revert selected files back to a previous state.
  - Revert the entire project back to a previous state.



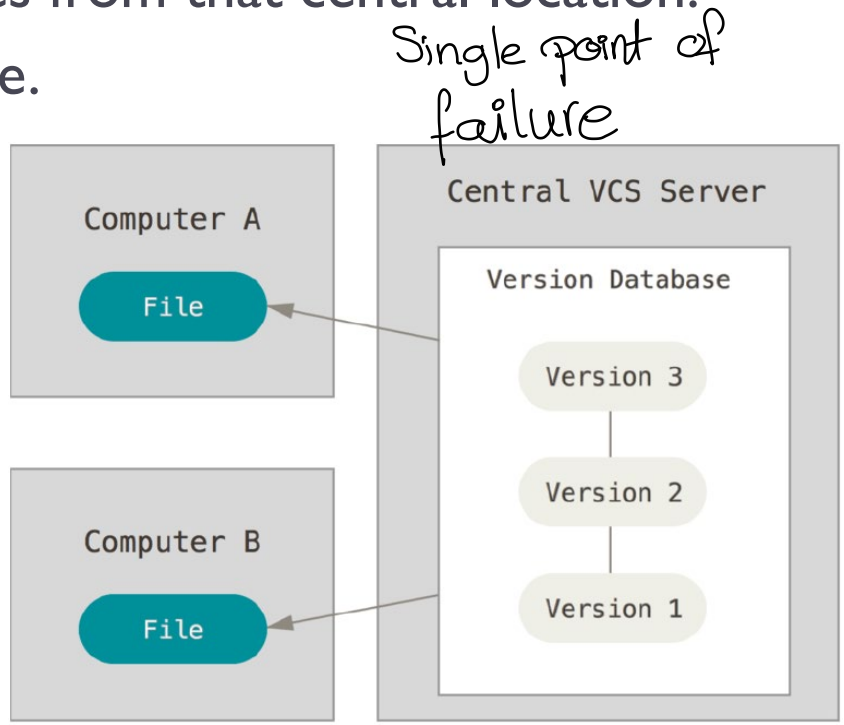
# Version Control (cont.)

- Compares changes over time:
  - See who last modified something that might be causing a problem.
  - See who introduced an issue and when.
  - Much more.
- **Local VCS**: → no collaboration!
  - Consists of a simple database that kept all the changes to files under revision control.
  - E.g., RCS (Revision Control System).
  - A local VCS does not allow collaboration with developers on other systems.



# Version Control (cont.)

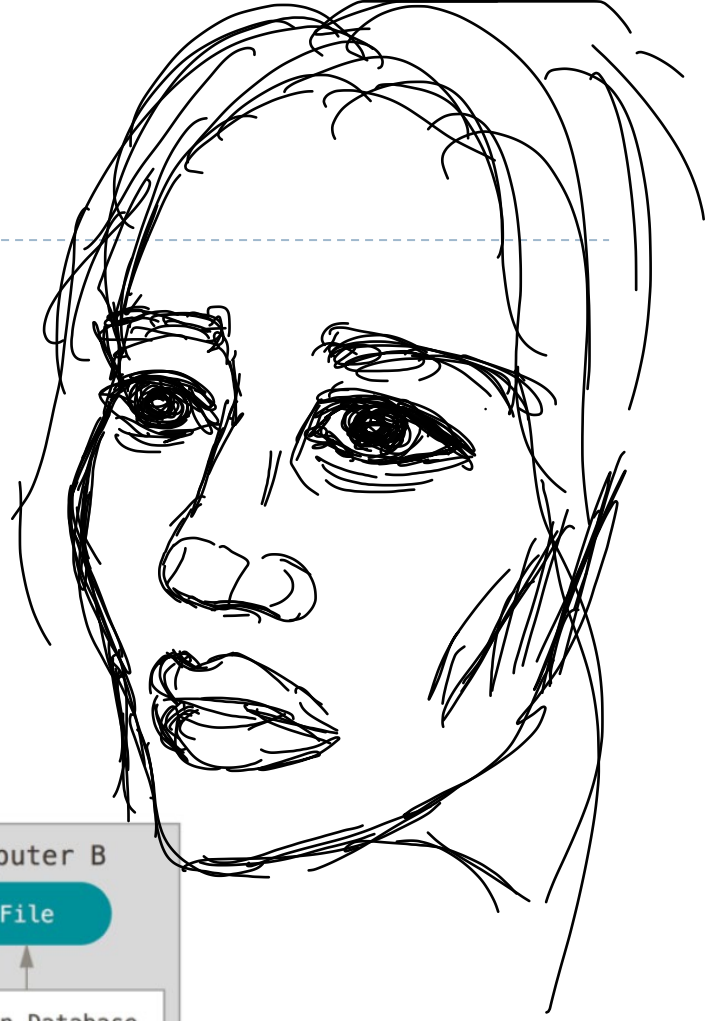
- **Centralised VCS**:
  - Overcomes the problem associated with local VCS.
  - Consists of a single server that contains all the versioned files.
  - Multiple clients can check out files from that central location.
  - E.g., CVS, Subversion and Perforce.
  - Problems:
    - Single point of failure.
    - Corruption of central database.

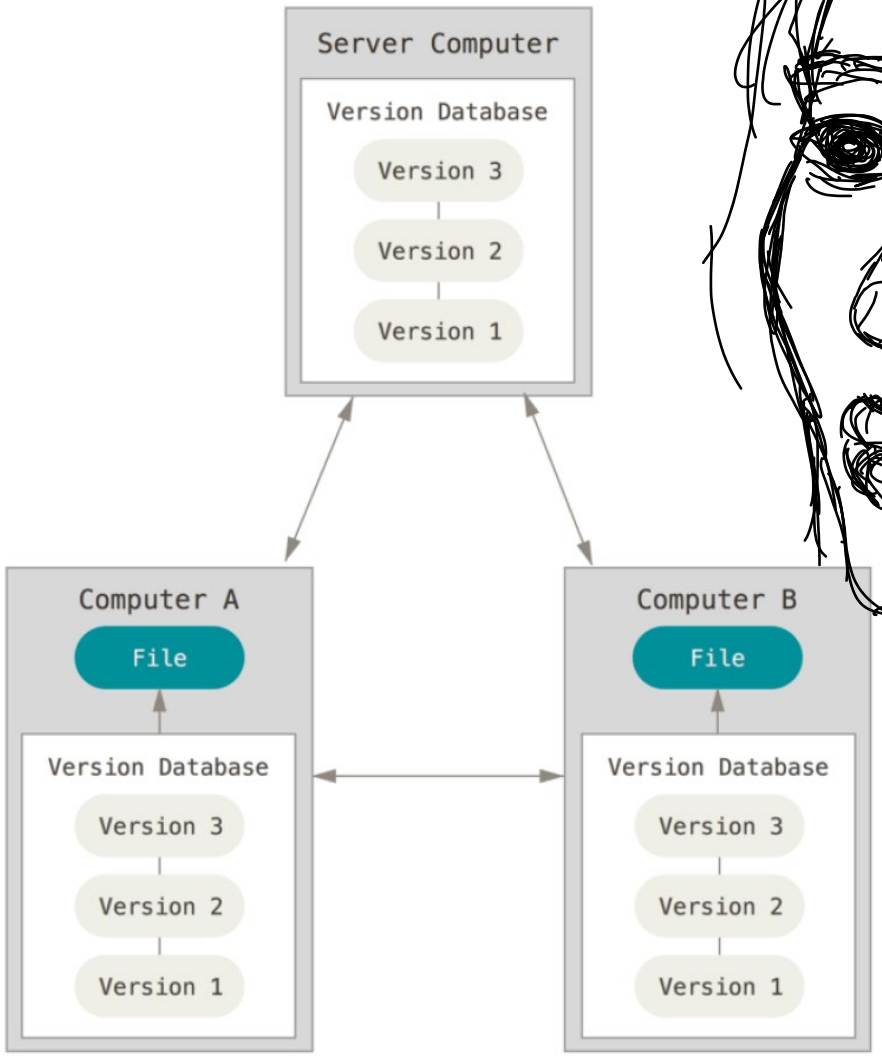


# Version Control (cont.)

- **Distributed VCS**:
  - Clients check out the latest snapshot of the files together with a full mirror of the repository.
  - The mirror or local repository includes the full history of the main remote repository.
  - E.g., Git and Mercurial.
  - Advantages:
    - Mitigates the single point of failure problem with centralised VCS.
    - Can work with multiple remote repositories.
    - Enables continuous development even if server is uncontactable.
    - Enables collaboration with different groups of people in different ways simultaneously within the same project.

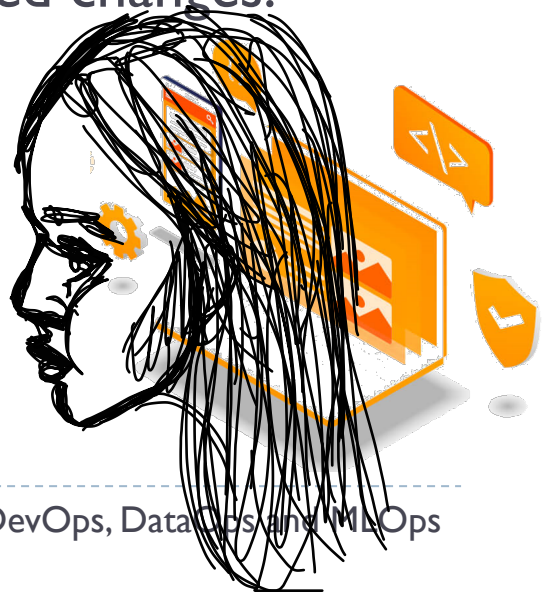
# Version Control (cont.)





# Continuous Monitoring and Logging

- What is **continuous monitoring and logging**?
  - Monitoring metrics and logs of application and infrastructure.
  - Assess how performance impacts the experience of product's end user.
- Objectives of continuous monitoring and logging:
  - Understand how changes or updates impact users.
  - Identify root causes of problems or unexpected changes.
  - Active monitoring is increasingly important as services must be available 24/7 and update frequency increases.

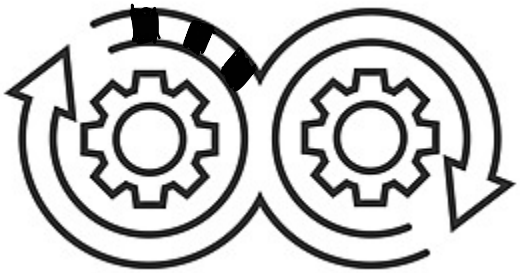


# Continuous Monitoring and Logging (cont.)

- How does continuous monitoring and logging work?
  - Also closely coupled to the infrastructure.
  - Increasingly use of cloud-based tools.
- What sort of problems/issues pertinent to analytics might be identified through monitoring and logging?

# DevOps Toolchain

- The DevOps methodology relies on effective tooling.
- What is **DevOps toolchain**?
  - Tools and technology that enable development and operations teams to collaborate across the entire software life cycle.
- Objectives of DevOps toolchain:
  - Automates manual tasks, manages complex environments at scale, and allows engineers to control the high velocity development process.
  - Help teams rapidly and reliably deploy and innovate for their customers.



# DevOps Toolchain (cont.)

- Major categories of DevOps tools:

| Category | Description | Examples |
|---|---|---|
| Project management tools | Enable teams to build a backlog of user stories that form coding projects, break them down into smaller tasks and track the tasks through to completion. | GitHub Issues and Jira. |
| Collaborative source code repositories | Version-controlled coding environments that let multiple developers work on the same code base. Code repositories should integrate with CI/CD, testing and other tools. | GitHub, GitLab and Bitbucket |
| CI/CD pipelines | Automate code checkout, building, testing and deployment. | Jenkins, CircleCI and Argo CD |
| Test automation frameworks | Software tools, libraries and best practices for automating unit, contract, functional, performance, usability, penetration and security tests. | Language specific tools and Selenium. |

we will do next week?

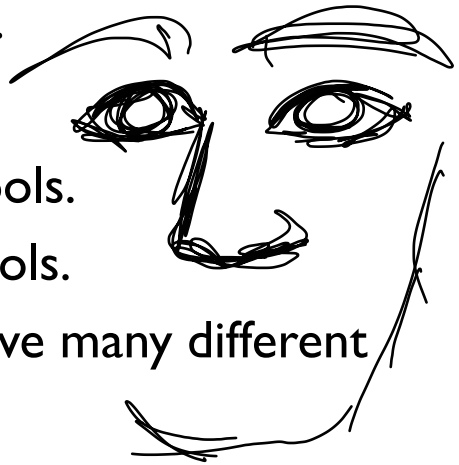
Python → PyTest.

# DevOps Toolchain (cont.)

| Category | Description | Examples |
|---|---|---|
| Configuration management (infrastructure as code) | Enable DevOps engineers to configure and provision fully versioned and fully documented infrastructure by executing a script. | Ansible, Chef, Puppet and Kubernetes (for containerised applications) |
| Monitoring tools | Identify and resolve system issues; gather and analyze data in real time to reveal how code changes impact application performance. | Datadog, Nagios, Prometheus and Splunk |
| Continuous feedback tools | Gather feedback from users, either through heatmapping (recording users' actions on screen), surveys, or self-service issue ticketing. | HotJar, Microsoft Clarity, TestFlight |

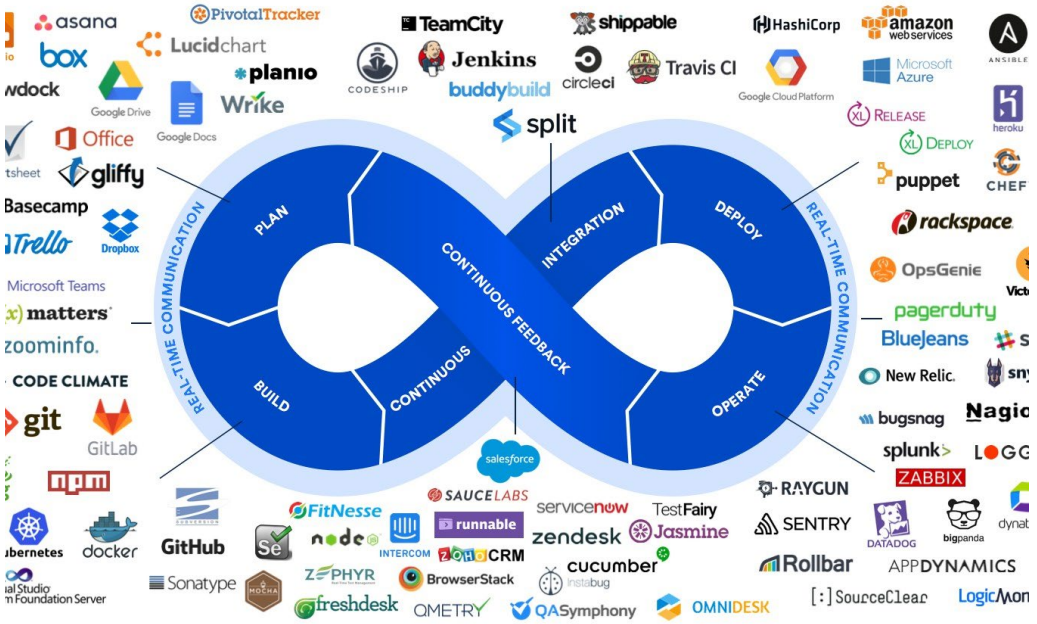
# DevOps Toolchain (cont.)

- Options for building DevOps toolchain:
  - **All-in-one toolchain**:
    - Provides a complete solution that may not integrate with other third-party tools.
    - Advantages – Useful for newcomers or quick start.
    - Disadvantages – Most established teams already have a set of tools they use and prefer, which may not integrate with a complete solution.
    - Mostly cloud-based, e.g., AWS DevOps services.
  - **Customisable toolchain**:
    - Customised for a team's needs with different tools.
    - Advantages – Allows teams to retain existing tools.
    - Disadvantages – One development stack can have many different customizable toolchain.



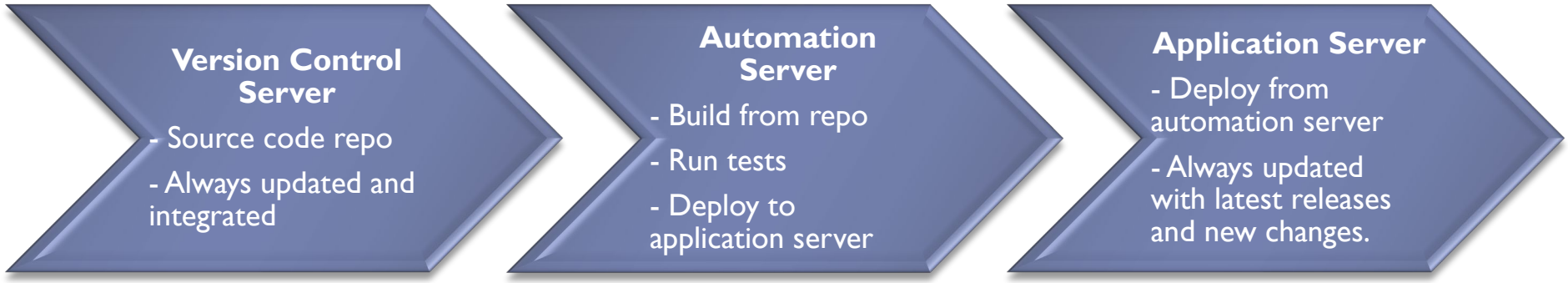
# DevOps Toolchain (cont.)

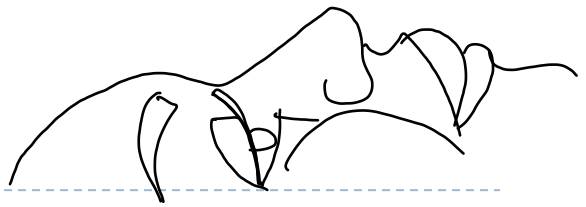
- Example of automated testing:
  - Just for Python – unittest, Pytest, PyUnit
- A more complete example of customisable toolchain:
  - Jira for planning and workflow tracking.
  - Kubernetes to provision individual development environments.
  - Github for collaborative coding.
  - Jenkins for CI/CD.



# DevOps Toolchain (cont.)

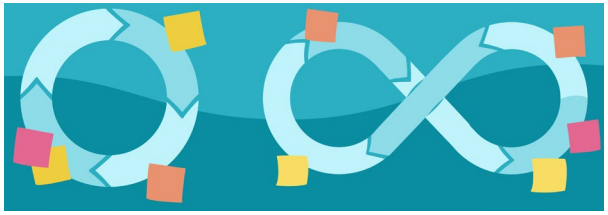
- A high-level overview to summarise the key toolchain elements:
  - DevOps relies on version control as the basis for tracking and integrating changes.



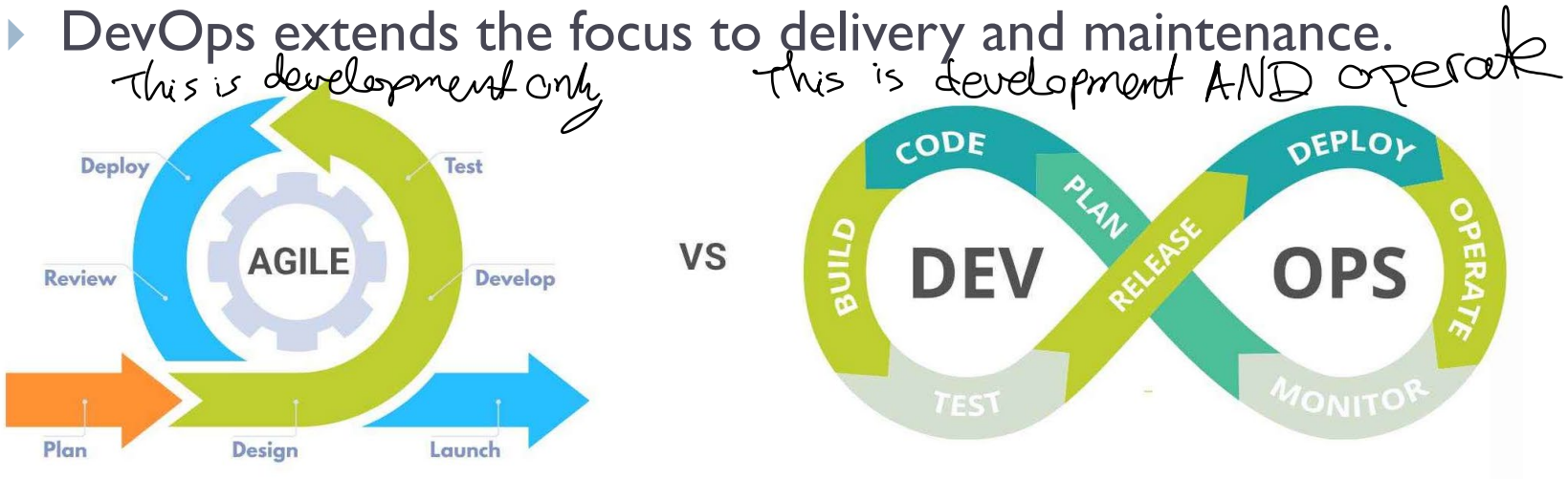
# Relationship between Agile and DevOps

- Quick recap of **agile** methodologies:
  - Agile emphasizes continuous cycles of incremental and iterative development process.
  - Address the shortfalls of the waterfall model.
  - Agile methodologies such as Scrum and Extreme Programming cover both managerial and technical issues of development.
- **DevOps** methodology:
  - Enables teams to build, test and release software faster and more reliably by incorporating agile principles and practices.
  - Emphasizes integration between development and operation.



# Relationship between Agile and DevOps (cont.)

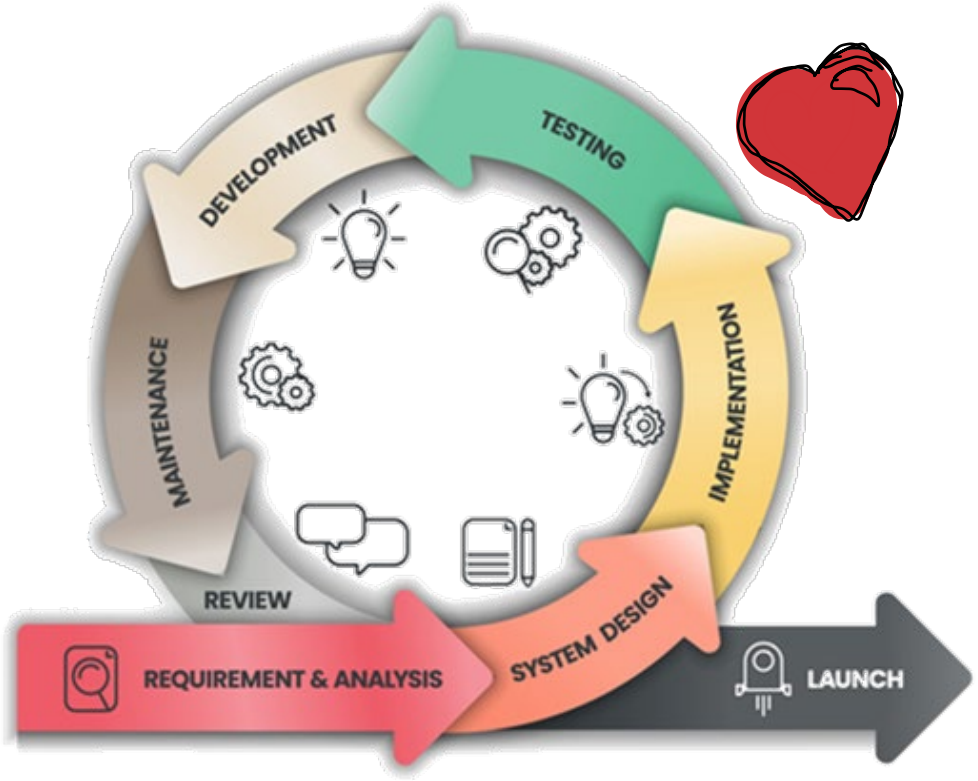
- Agile and DevOps are synergistic:
  - Agile centers the flow of software from ideation to code completion.
  - DevOps extends the focus to delivery and maintenance.

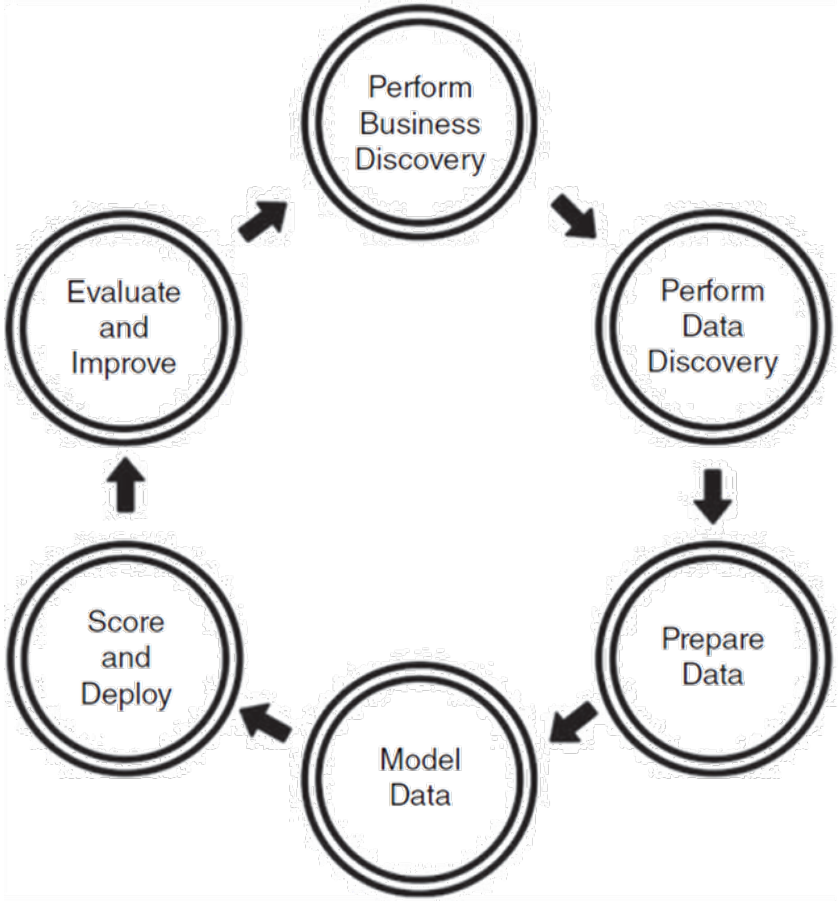



- Goals of agile and DevOps are the same:
  - Improve the speed and quality of software development.
  - Makes very little sense to talk about one without the other

# Recap of Agile Analytics

- Agile analytics is similar to agile software development:

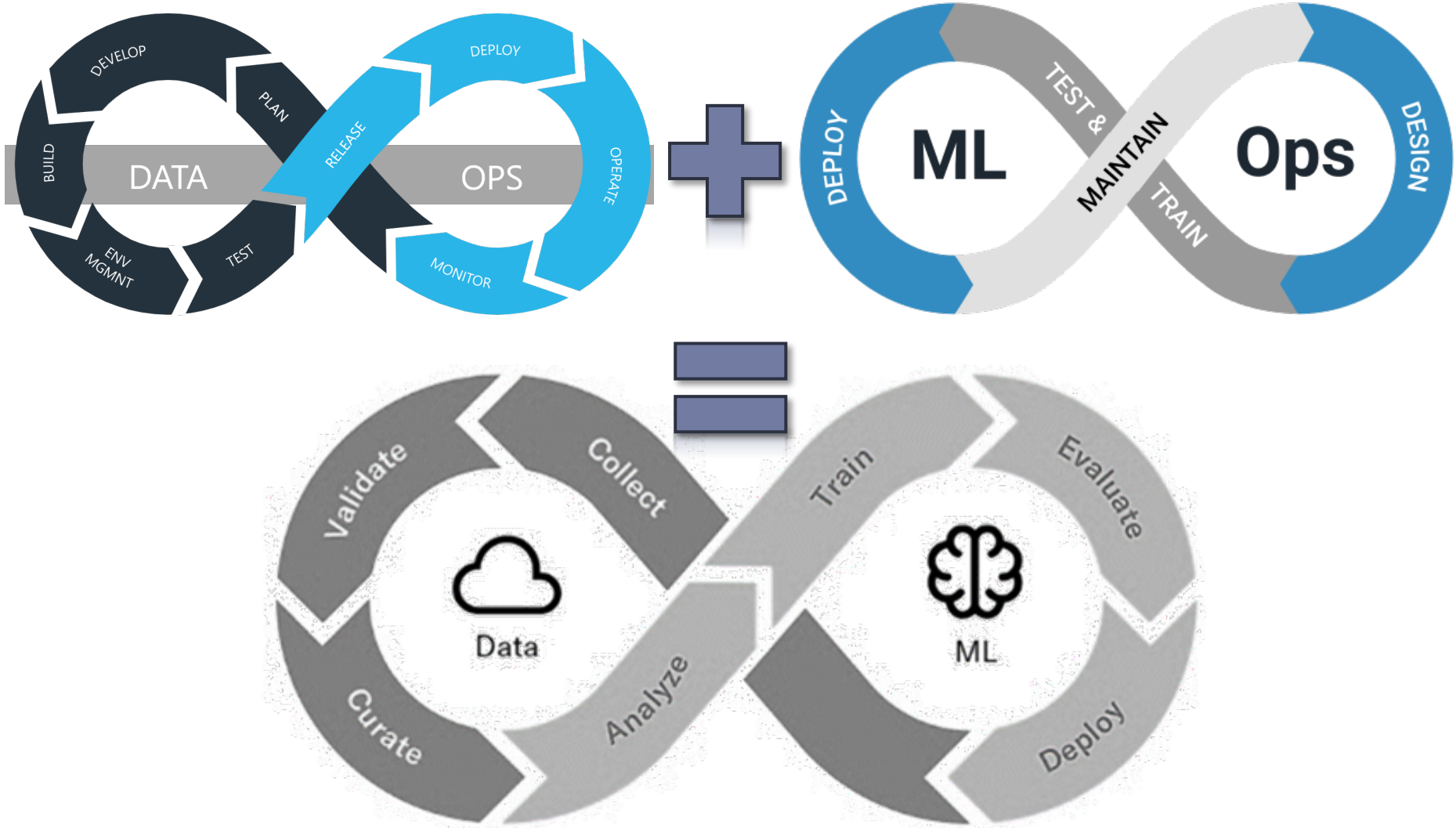


# DevOps, Agile Software Development and Agile Analytics

- The deliverable or output aren't that different either:
  - Full-code approach of analytics modelling remain the preferred option providing best performance and greatest flexibility.
  - The output of analytics is mainly a model but typically also include software applications.
  - Agile analytics and DevOps are synergistic too.
- But important difference is that analytics has a (not so) clear distinction between data and model:
  - Ultimately, data needs to be connected to model.
  - **DataOps** – Automate data integration and transformation.
  - **MLOps** – Automate IT operation and infrastructure involve in developing machine learning models.

# DevOps, Agile Software Development and Agile Analytics (cont.)



# Overview of MLOps

- What is **MLOps**?
  - MLOps is a methodology to standardise and streamline machine learning life cycle.
  - The life cycle looks relatively simple.
  - But managing this life cycle at scale in real-world organisational context is challenging.

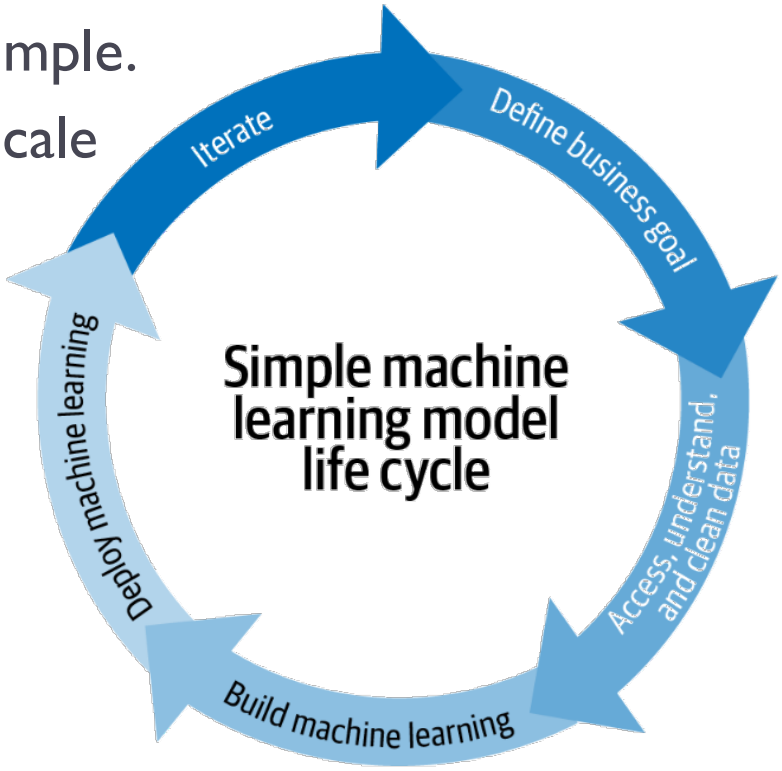



*Figure 1-2. A simple representation of the machine learning model life cycle, which often underplays the need for MLOps, compared to Figure 1-3*

# Overview of MLOps (cont.)

- There are many real-world complexities of machine learning life cycle.
- **Too many dependencies**:
  - Data is constantly changing and so are business needs.
  - Results need to be relayed back to the business to ensure that the reality of the model in production and on production data aligns with expectation.
- **Not everyone speaks the same language**:
  - The life cycle involves stakeholders from business, data science and IT teams.
  - But none of these groups are using the same tools or share common skills to serve as the basis for communication.

# Overview of MLOps (cont.)

- **Data scientist are not software engineers**:
  - Data scientists are specialists at building models and not developing software applications.
  - But many data scientists found themselves having to take on multiple roles.
  - Turnover in staffs further complicates things.
- DevOps aims to integrate development team and IT operations team.
- MLOps aims to integrate all stakeholders involved in the machine learning life cycle.

# Overview of MLOps (cont.)

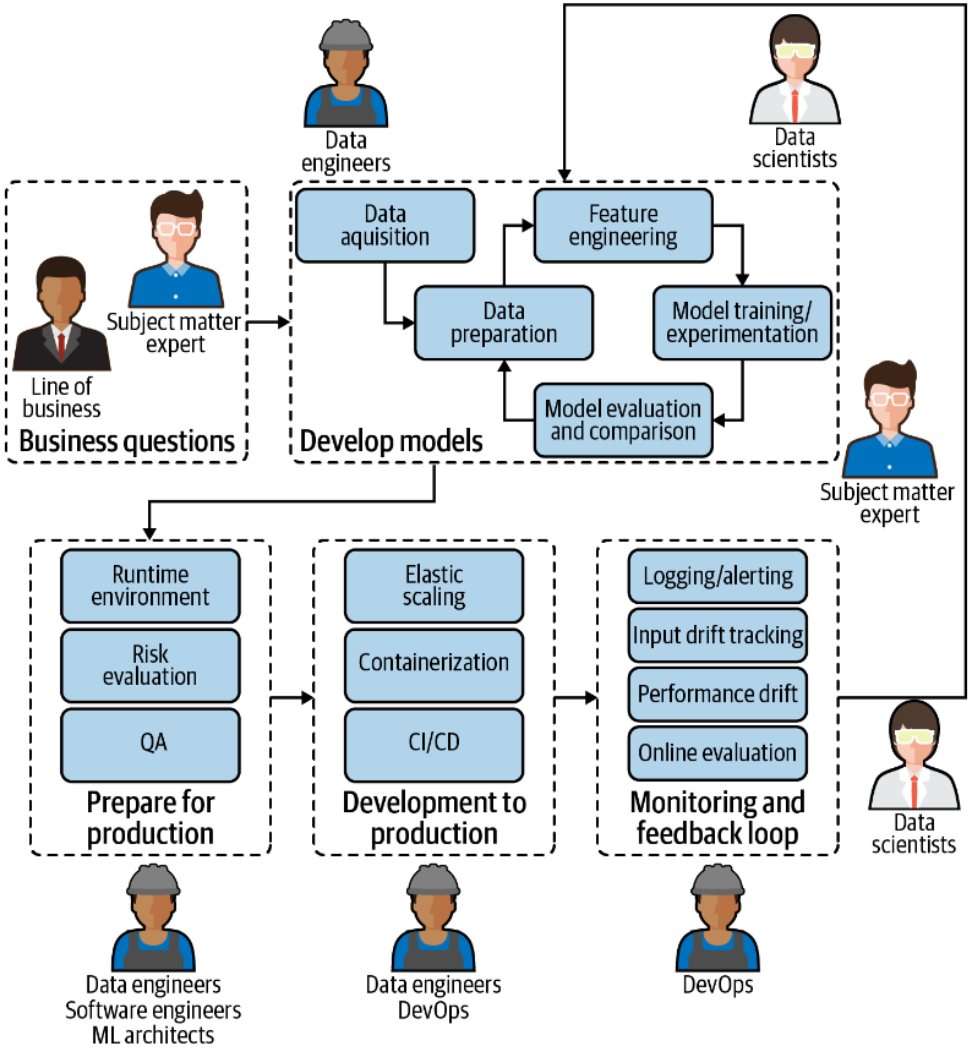



Figure 1-3. The realistic picture of a machine learning model life cycle inside an average organization today, which involves many different people with completely different skill sets and who are often using entirely different tools.

# Overview of DataOps

- MLOps is further complicated by DataOps.
- What is **DataOps**?
  - Focus on the data life cycle to enhance data quality and metadata management.
  - Provide business-ready data that is quickly available for use.
- A typical scenario that necessitates DataOps:
  - There is a sudden change in data that a model relies on.
  - DataOps would alert the business team to deal more carefully with the latest insights.
  - Data team would be notified to investigate the change and rectify the associated problem.

# Overview of DataOps (cont.)

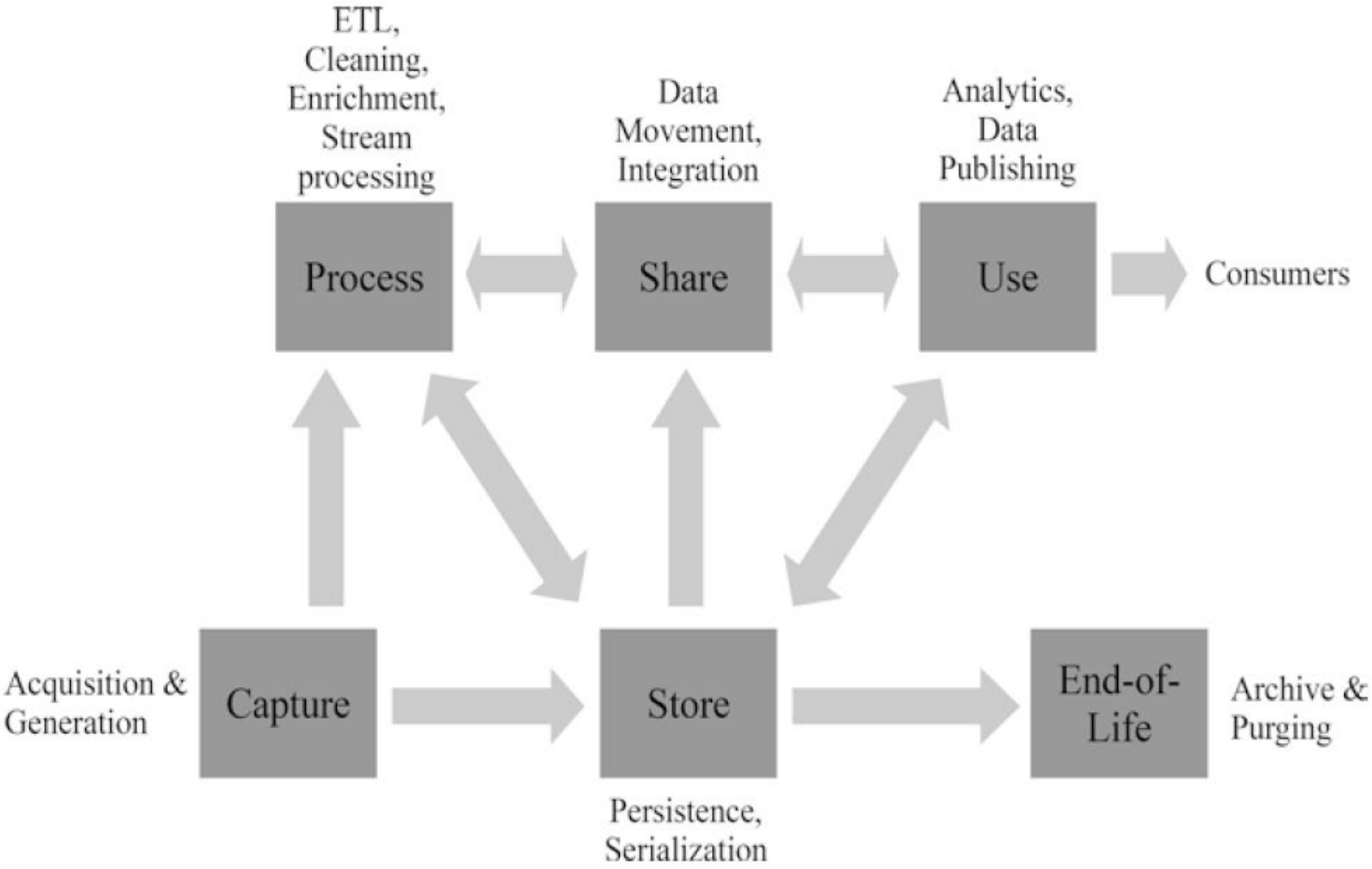



***Figure 2-1.*** *The stages of the data lifecycle*

# Overview of DataOps (cont.)

- Data is no longer just about IT:
  - Data is now a critical asset for analytical decision-making long after operational business processes have ended.
  - Data is the new oil, the source for competitive advantages.
  - Data and analytical output need not be perfect but must achieve an acceptable level of accuracy.
- DataOps aims to integrate stakeholders involved in the data life cycle and beyond.

# Summary

- DevOps is a methodology for improving the agility of software application development by integrating development team with IT operations team.
- DevOps entails cultural philosophies, practices and tools.
- Important DevOps practices include frequent but small updates, microservices architecture and CI/CD.
- DevOps toolchains are inherently complex due to the variety of tools involved.

# Q&A

# Next Lecture…


- Learn about:
  - DataOps concepts.