Bridging the Gap:
From Data Science to Production

Florian Wilhelm

EuroPython 2018 @ Edinburgh, 2018-07-25

@FlorianWilhelm

FlorianWilhelm

florianwilhelm.info

Dr. Florian Wilhelm

Principal Data Scientist @ inovex

Special Interests

- Mathematical Modelling
- Recommendation Systems
- Data Science in Production
- Python Data Stack

inovex

IT-project house for digital transformation:

- Agile Development & Management
- Web · UI/UX · Replatforming · Microservices
- Mobile · Apps · Smart Devices · Robotics
- Big Data & Business Intelligence Platforms
- Data Science · Data Products · Search · Deep Learning
- Data Center Automation · DevOps · Cloud · Hosting
- Trainings & Coachings

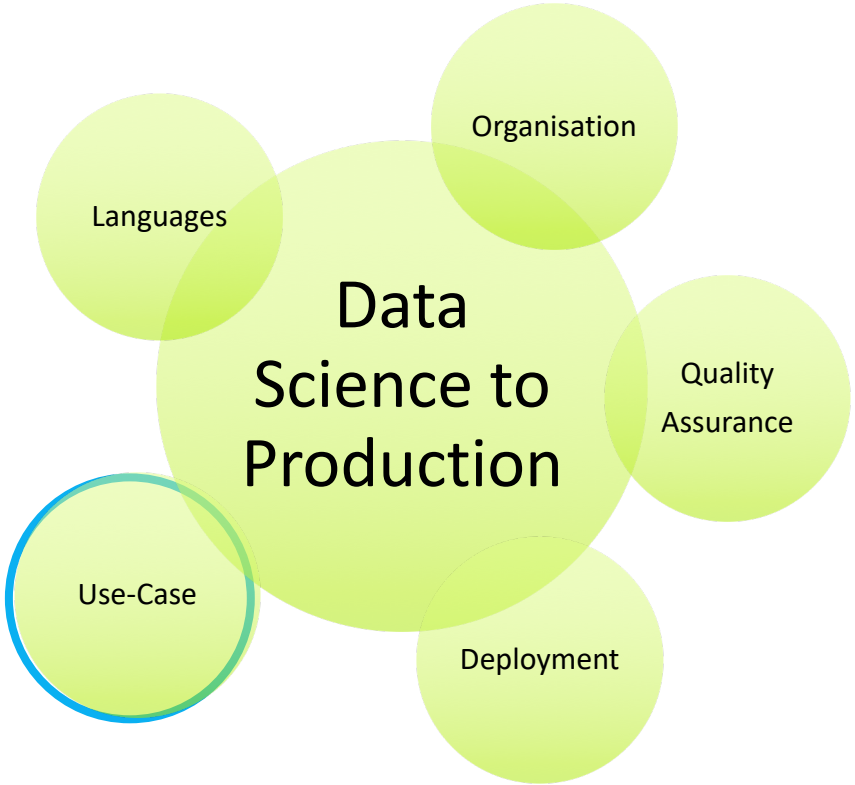
inovex offices in
Karlsruhe · Cologne · Munich · Pforzheim · Hamburg · Stuttgart.

www.inovex.de

Using technology to inspire our clients. *And ourselves*.

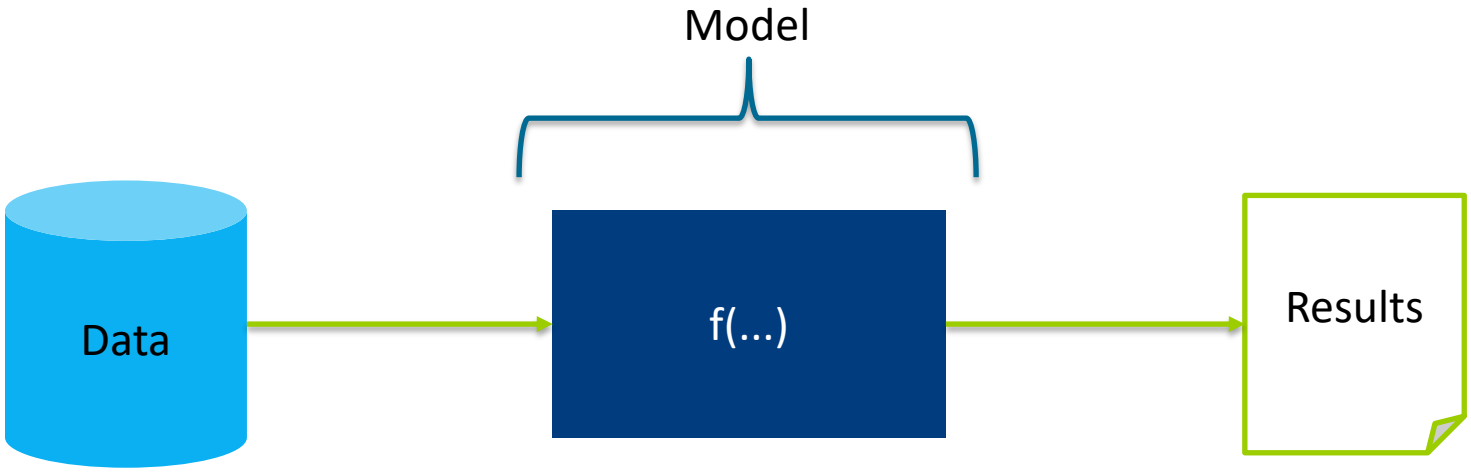
Agenda

Many facets



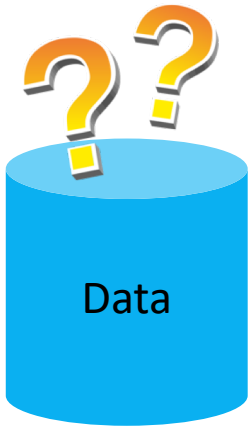
Use-Case: High Level Perspective

What does your model pipeline look like?



Use-Case: High Level Perspective

What is your Data Source?



Variants:

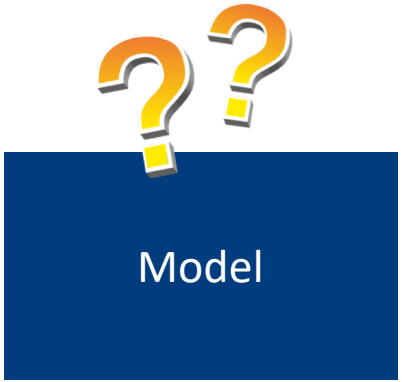
- Database (PostgreSQL, C*)
- Distributed Filesystem (HDFS)
- Stream (Kafka)
- ...

How is your data accessed?

What are the frequency and recency requirements?
Batch, Near-Realtime, Realtime, Stream?

Use-Case: High Level Perspective

What is a model?



Model includes:

- Preprocessing (cleansing, imputation, scaling)
- Construction of derived features (EMAs)
- Machine Learning Algorithm (Random Forest, ANN)
- …

Is the input of your model raw data or pregenerated features?

Does your model have a state?

Use-Case: High Level Perspective

How is your result stored?

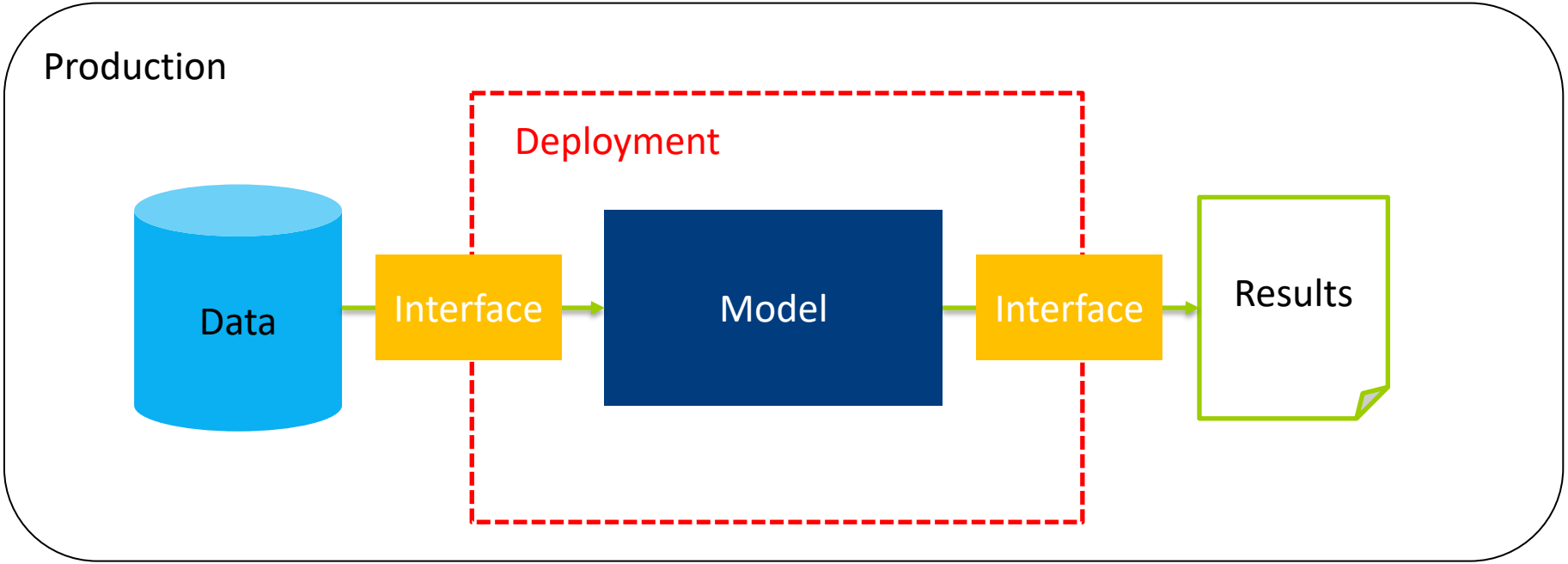
Variants:

- Database (PostgreSQL, C*)
- Distributed Filesystem (HDFS)
- Stream (Kafka)
- On demand (REST API)
- …

What are the frequency and recency requirements?
Batch, Near-Realtime, Realtime, Stream?

Use-Case: High Level Perspective

Our challenge



Use-Case Evaluation

Characteristics of a Data Use-case

Delivery	Problem Class	Volume & Velocity	Inference / Prediction	Technical Conditions
WebService	Classification	10 GB weekly	Batch	Java-Stack + Python
Stream	Regression	1 GB daily	Near-Realtime	On-Premise
Database	Recommendation	10k events/s	Realtime	AWS Cloud
	Explainability?		Stream	

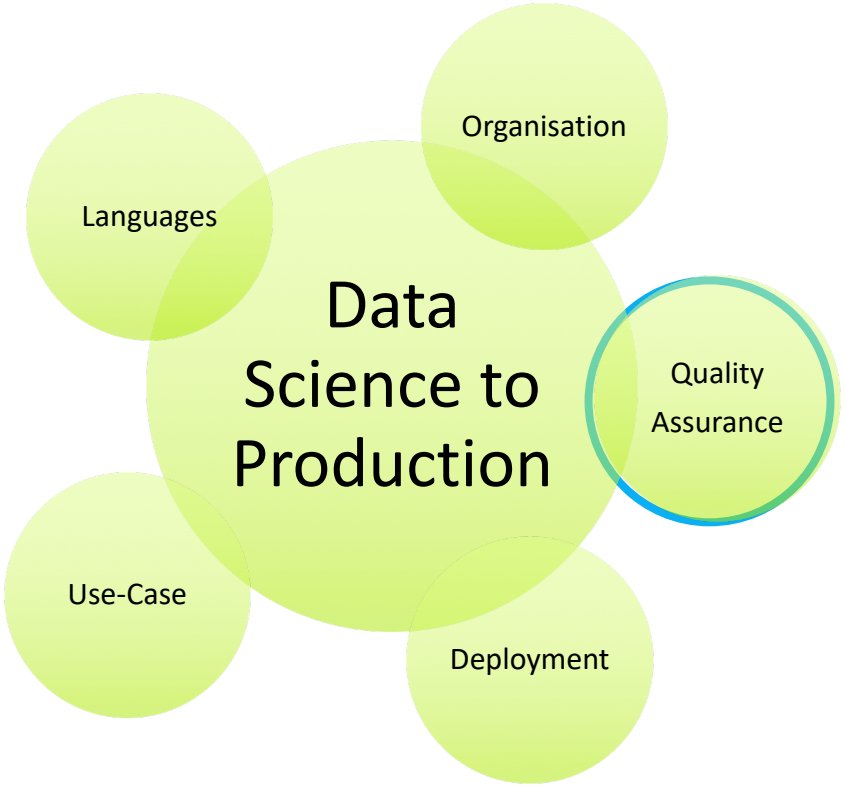
Note down your specific requirements before thinking about an architecture.
There is no one size fits all!

Use-Case: High Level Perspective

Right from the Start

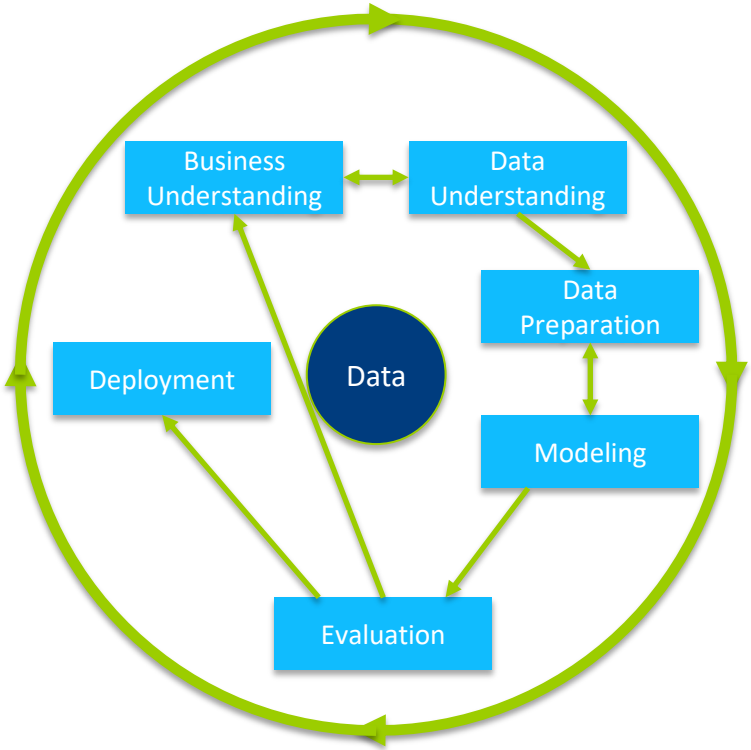
- State the requirements of your data use-case
- Identify and check data sources
- Define interfaces with other teams/departments
- Test the whole data flow and infrastructure early on with a dummy model

Many facets


Organisation
Languages
Data Science to Production
Quality Assurance
Use-Case
Deployment

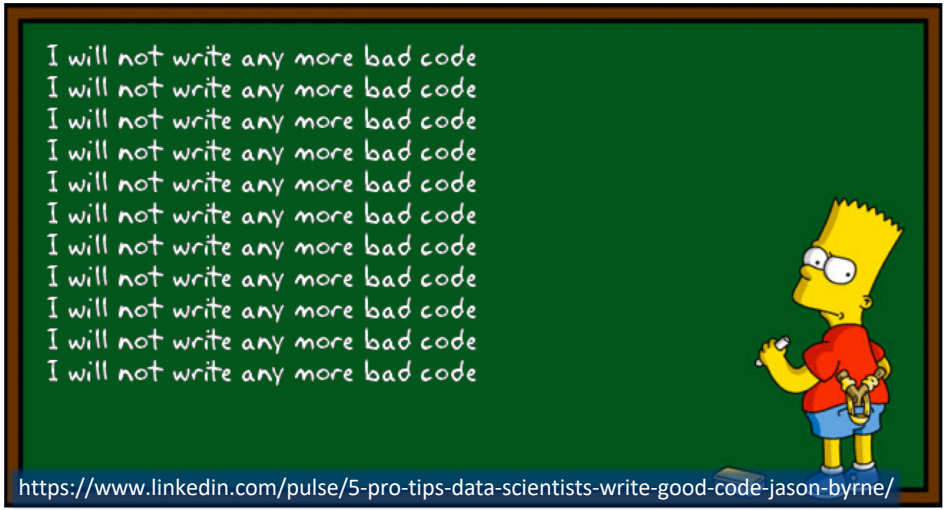
It‘s an iterative Process

Quality Assurance for smooth iterations



CRISP-DM

Clean Code



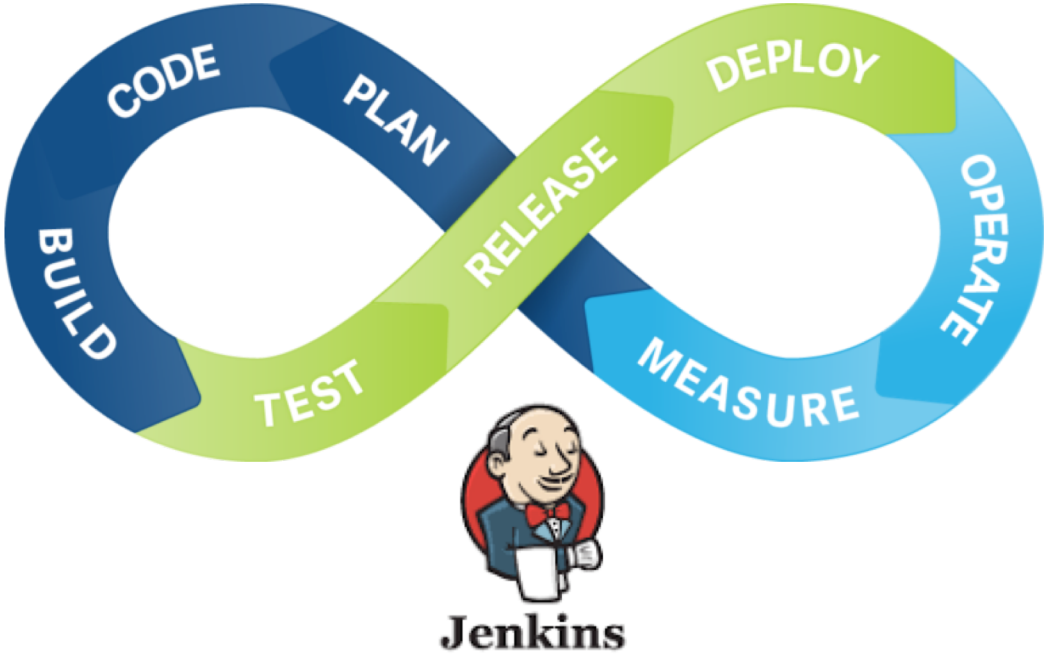
Resources:

- Software Design Patterns
- SOLID Principles
- The Pragmatic Programmer
- The Software Craftsman

Clean code is code that is easy to understand and easy to change.

Continuous Integration

- Version, package and manage your artefacts
- Provide tests (unit, systems, ...)
- Automize as much as possible
- Embrace processes



Monitoring

KPI and Stats

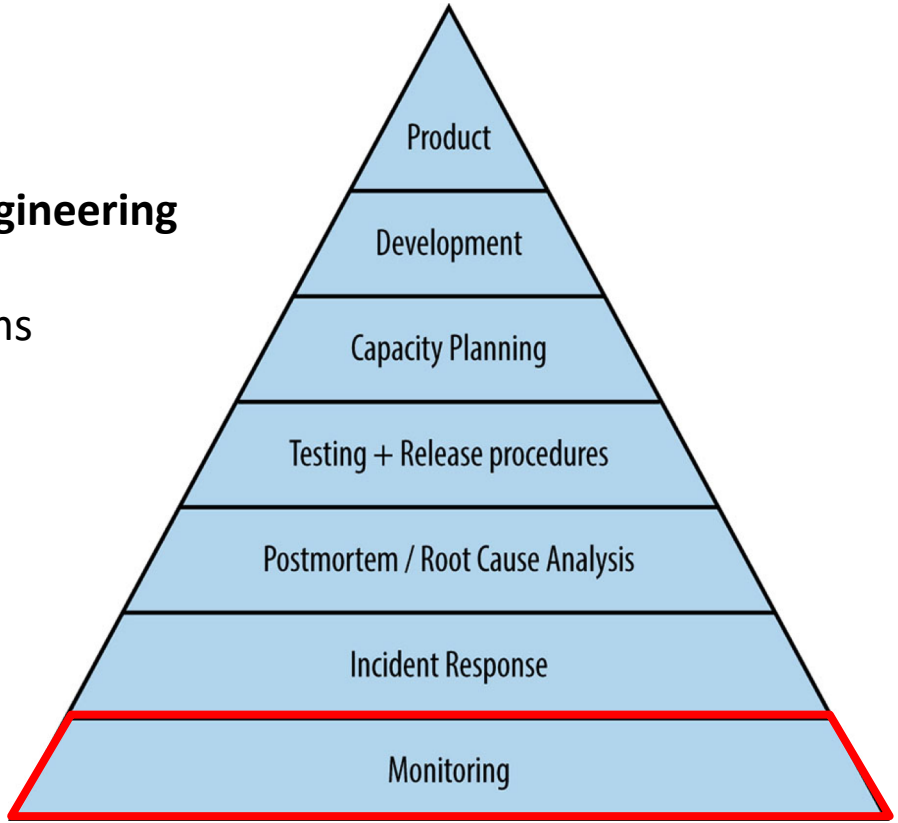
- KPIs (CTR, Conversions)
- Number of requests
- Timeouts, delays
- Total number of predictions
- Runtimes
- …

All monitoring needs to be linked to the currently running version of your model!

Monitoring

@Google

Site Reliability Engineering
How Google Runs
Production Systems



Monitoring

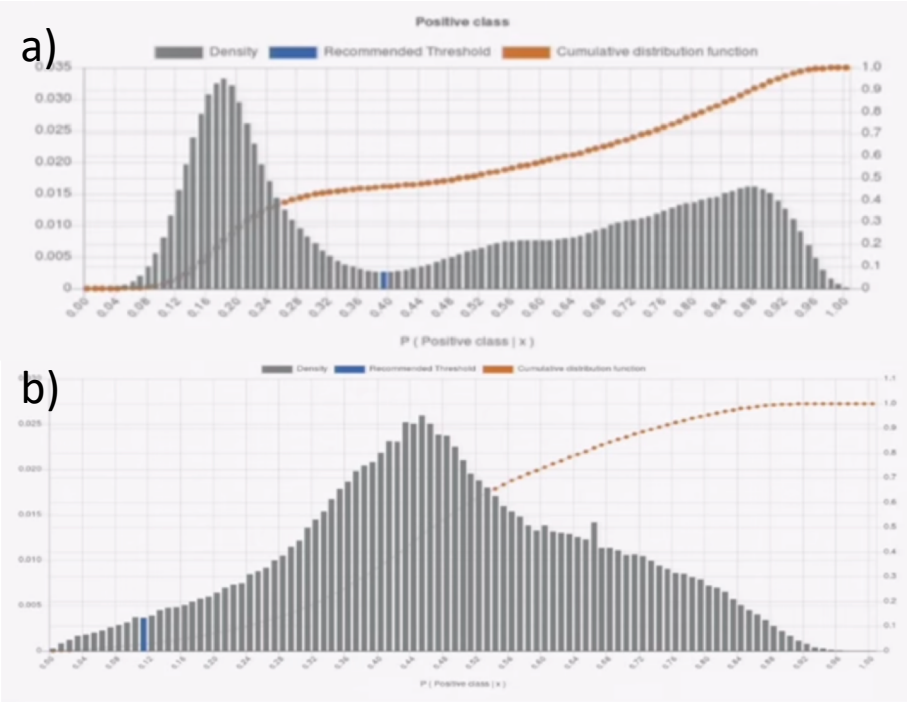
Model Stats

Monitor the results of model‘s predicitons

Example:

Response Distribution Analysis

a) working model
b) confused model



A/B Tests

Feedback for your model

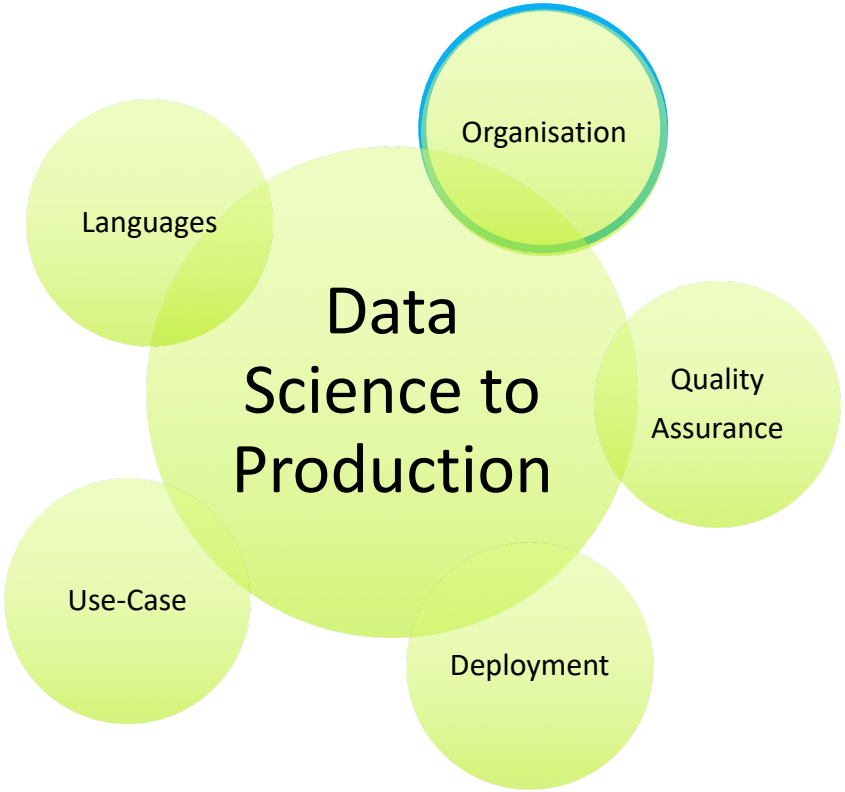
- Always compare your “improved“ model to the current baseline
- Allows comparing two models not only in offline metrics but also online metrics and KPIs.
- Also possible to adjust hyperparameters with online feedback, e.g. multi-armed bandit

A/B Tests

Technical requirements

- Versioning of your models to allow linking them to test groups
- Deploying and serving several models at the same time independently (needed for fast rollback anyway!)
- Tracking the results of a given model up to the point of facing the customer

Many facets

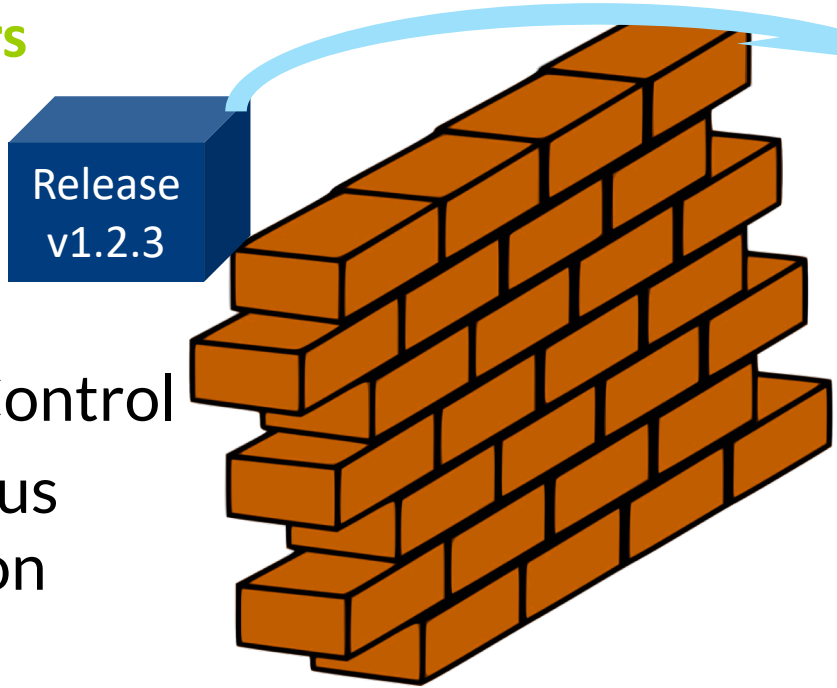

Organisation
Languages
Data Science to Production
Quality Assurance
Use-Case
Deployment

Organisation of Teams

Wall of Confusion

Developers

- Code
- Tests
- Releases
- Version Control
- Continuous Integration
- Features



Operations

- Packaging
- Deployment
- Lifecycle
- Configuration
- Security
- Monitoring

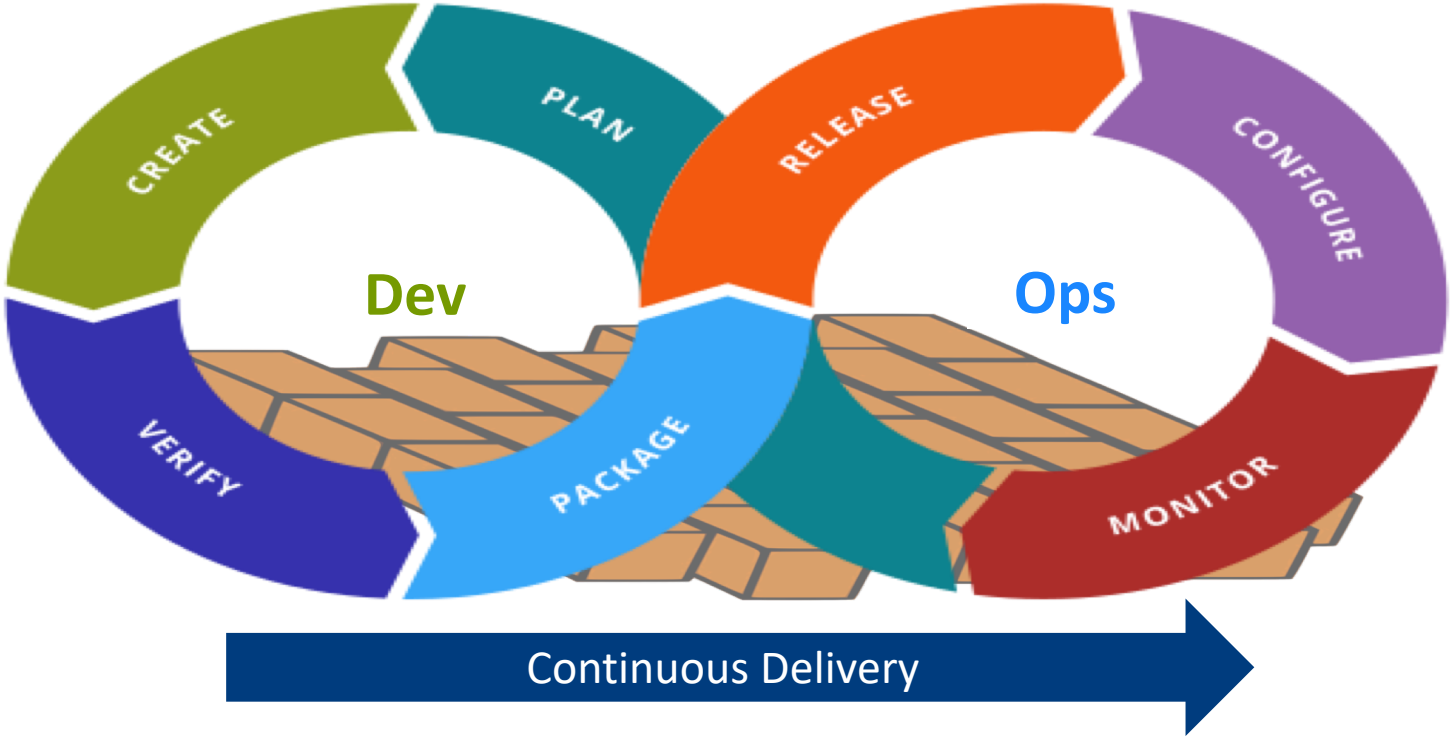
Different Cultures/Thinking

Wrong Approach!

- Especially dangerous separation for data products/features
- Speed and Time to Market are important thus “not my job“-thinking hurts
- “I made a great model“ vs. „We made a great data product“

Organisation of Teams

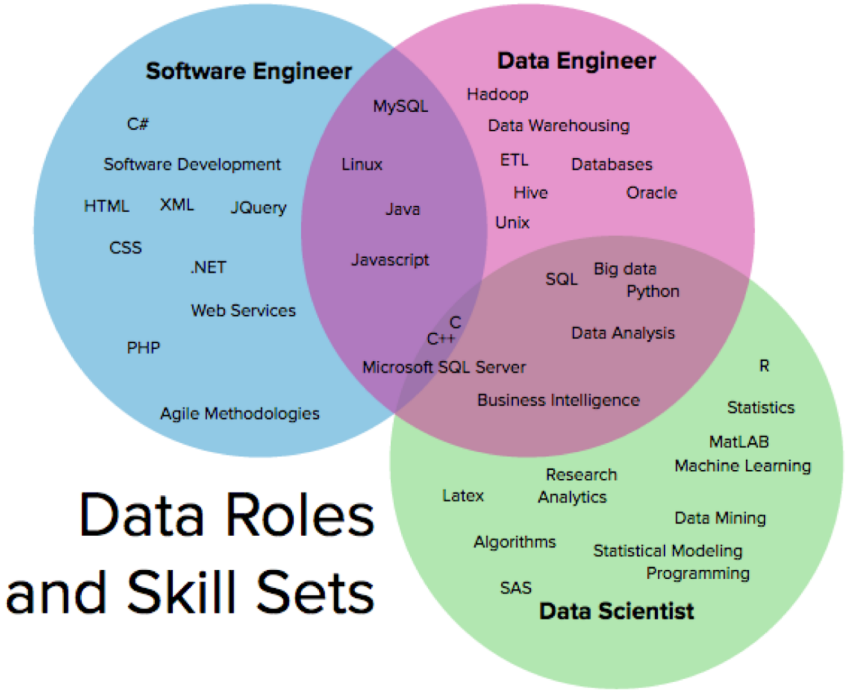
Overcoming the Wall of Confusion



Heterogeneous Teams

How to bring Data Scientists into DevOps?

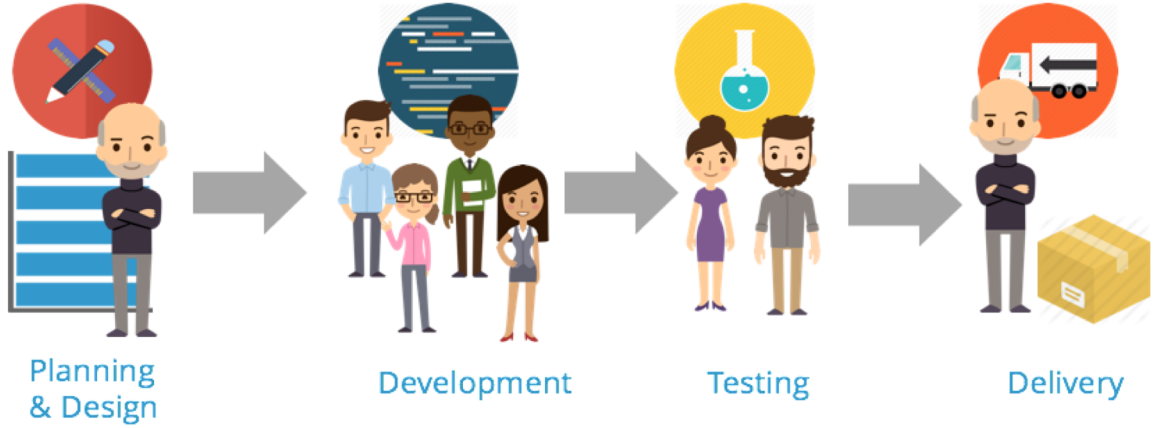
- Pure teams of Data Scientists often struggle to get anything in production
- As a minimum complement, SW and Data Engineers are needed. (2-3 Engineers per Data Scientist)
- Optionally a Data Product Manager as well as an UI/UX expert if necessary



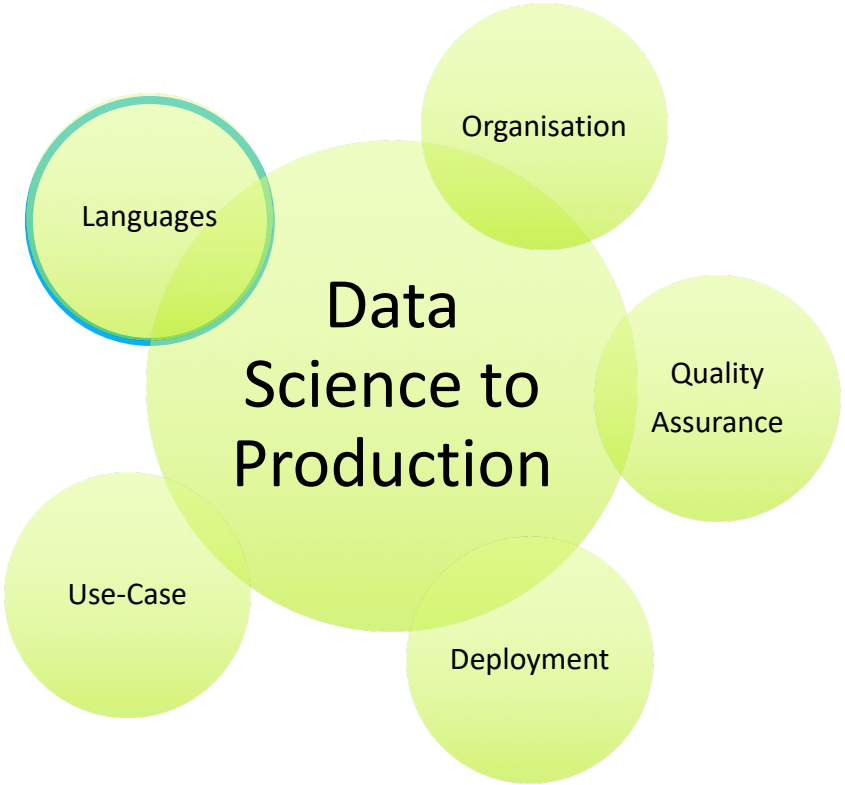
Organisation around Features

Responsibility with vertical teams

- Fully autonomous teams
- End-to-end responsibility for a feature
- Works well with Agile Methods like Scrum
- Faster delivery and less politics



Many facets


Organisation
Languages
Data Science to Production
Quality Assurance
Use-Case
Deployment

Programming Languages

The Two Language Problem

Industry

- Java stack common also Scala
- Strongly typed
- Emphasize on robustness and edge cases
- Industrial standards for deployment

Science

- Often Python and R
- Dynamic typed since easier to get the job done
- Emphasize on fancy methods and results
- Runs on my machine

Language Problem

Solution: Select one to rule them all!

- Having a single language reduces the complexity of deployment
- Implementation efforts due to abandoning one ecosystem totally

Language Problem

Solution: Python in production

- Especially easy for batch prediction use-cases
- If a web service is needed flask is a viable option
- Scale horizontally during prediction and use a big metal node for training a model
- Tap into the Hadoop world by using PySpark, PyHive etc.
- Consider isolated containers using docker

Language Problem

Solution: PoCs in Python/R, rewrite in Java for production

- Lots of efforts and slow
- Iterations and new feature are hard to implement
- Reproducability of bugs is cumbersome
- Pro: Everyone gets what they want

Worst-case Scenario

Language Problem

Solution: Exchangable formats

- Works great in theory
- Limited functionality and flexibility
- No guarantee the same model description will be interpreted the same by two different implementations
- Preprocessing / feature generation not included

Language Problem

Solution: Frameworks

- Various language bindings allow developing in Python/R and running on the Java stack
- Be aware if framework also covers feature generation
- Ease of use at the cost of flexibility

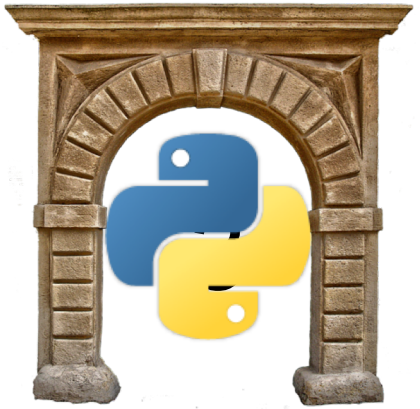
Two Language Problem

Three concepts of dealing with it

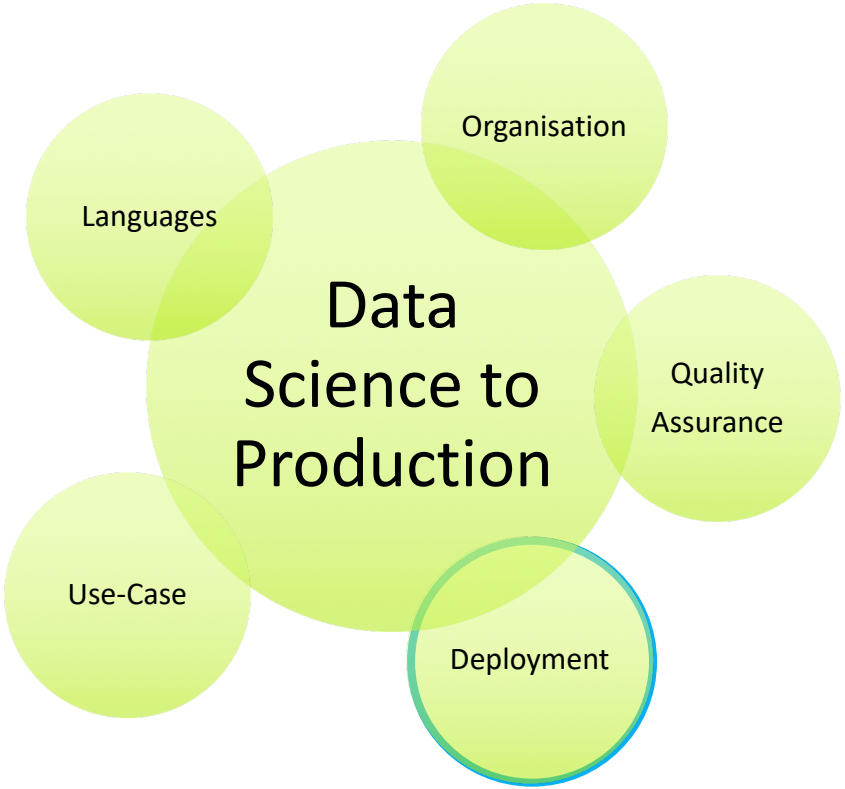
Reimplement

Frameworks

Single Language

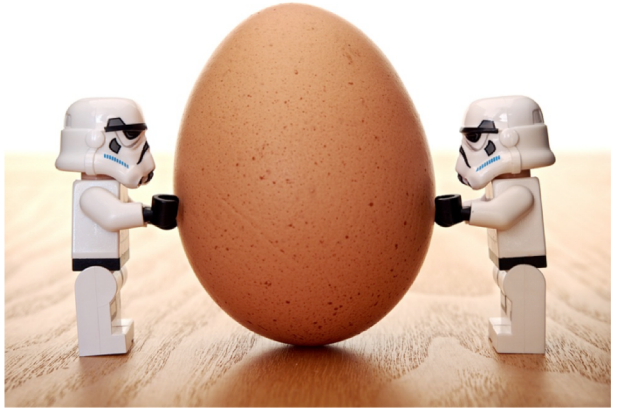


Many facets


Organisation
Languages
Data Science to Production
Quality Assurance
Use-Case
Deployment

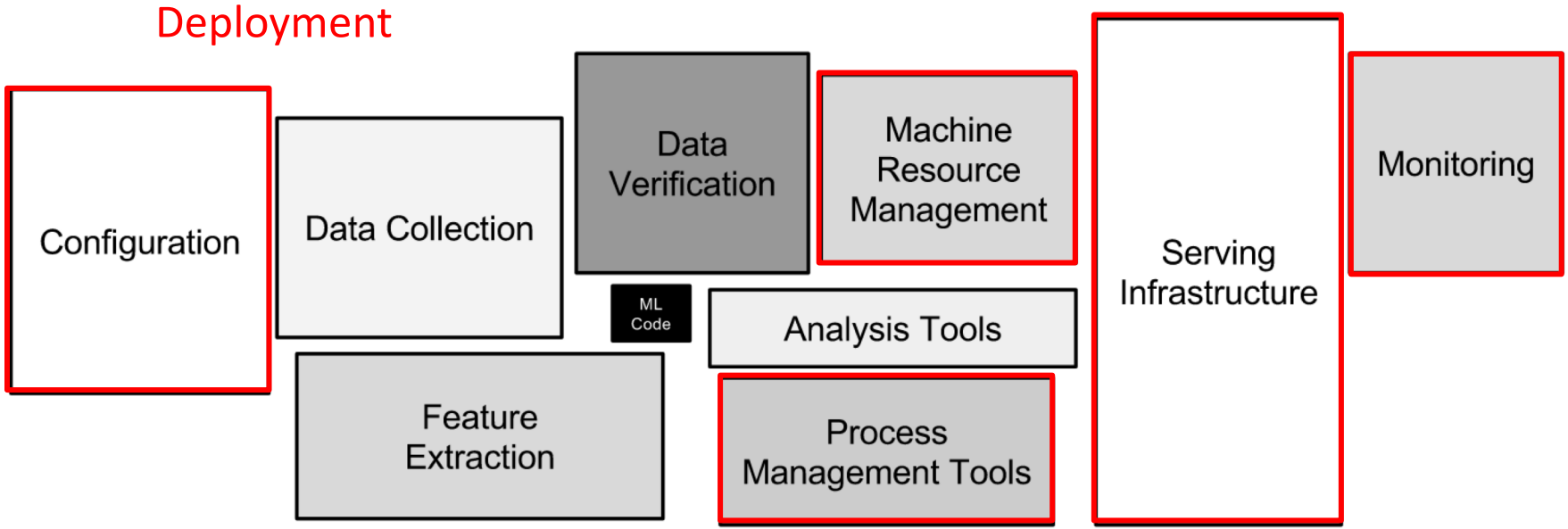
Deployment

Deployment heavily depends on the chosen approach!



Still some software engineering principles apply like Continuous Integration or even Continuous Delivery

Technical Debt in ML Pipelines



Deployment

General principles

- Versioning & packaging, defined processes, quality management
- Keep the development and production environment as similar as possible
- Automation is a must, avoid human error!
- Isolated and controllable environments are a great idea, e.g. Docker.

Google‘s Best Practices for ML Engineering

inovex

Best of Google‘s rules

Rule #1: Don’t be afraid to launch a product without machine learning.

Rule #2: First, design and implement metrics

Rule #4: Keep the first model simple and get the infrastructure right.

Rule #5: Test the infrastructure independently from the machine learning

Rule #9: Detect problems before exporting models.

Rule #11: Give feature columns owners and documentation.

Rule #13: Choose a simple, observable and attributable metric for your first objective.

Rule #14: Starting with an interpretable model makes debugging easier.

Rule #16: Plan to launch and iterate.

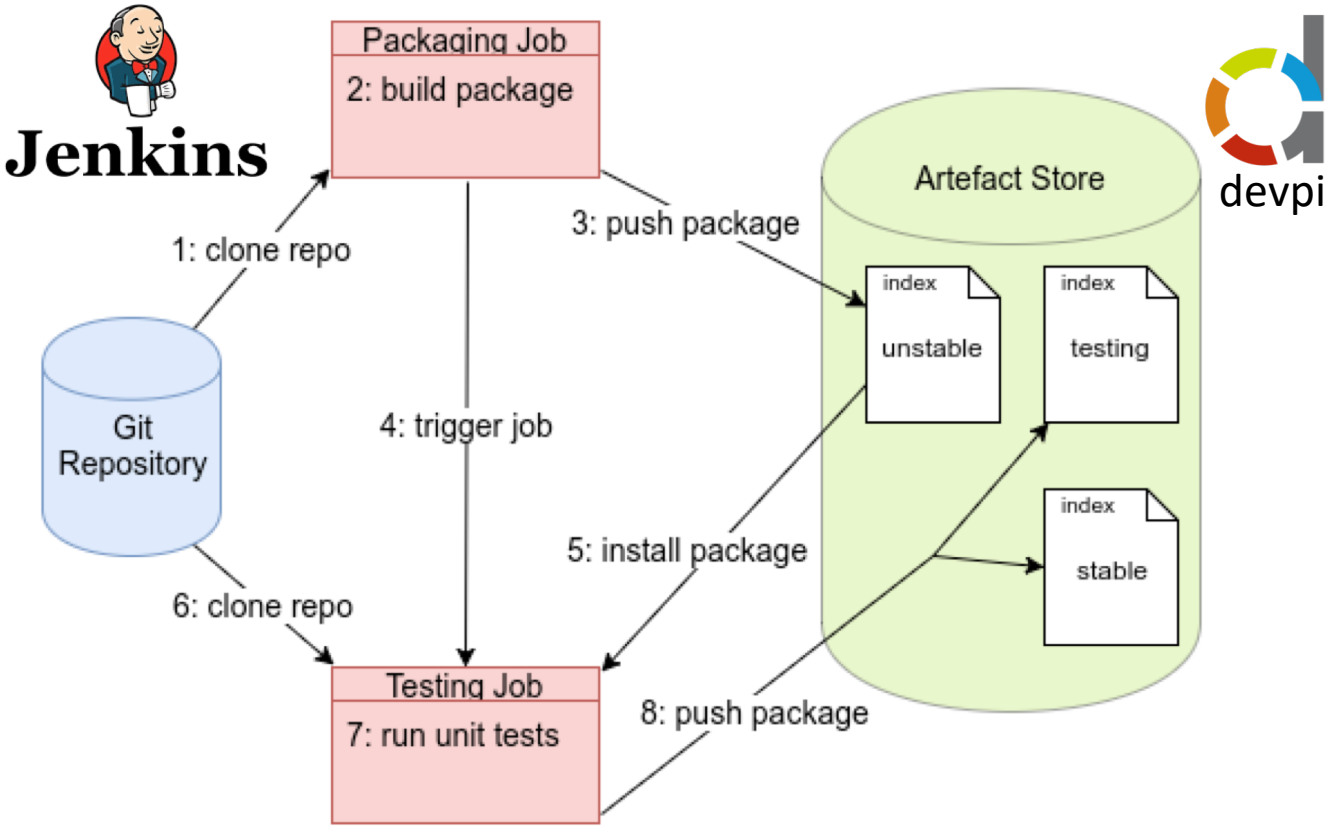
Rule #24: Measure the delta between models.

Rule #27: Try to quantify observed undesirable behavior. "**measure first, optimize second**“

Rule #32: Re-use code between your training pipeline and your serving pipeline whenever possible.

Most of the problems are engineering problems!

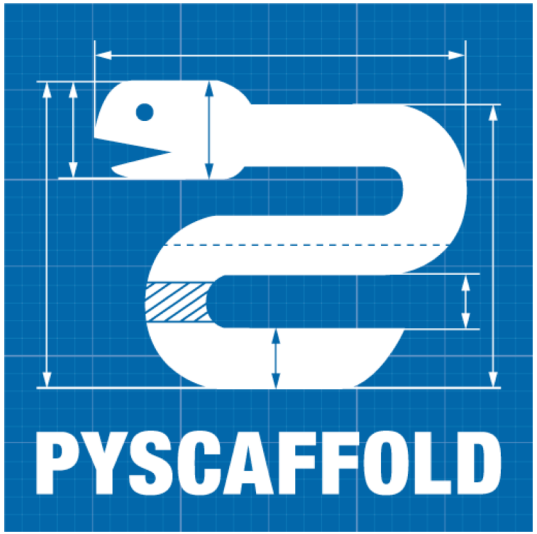
Example: Continuous Integration



Example: Python Package/Distribution

PyScaffold

- Easy and sane Python packaging
- Proper versioning of every commit
- Git integration, e.g. pre-commit
- Declarative configuration with setup.cfg
- Follows community standards
- Many extensions available

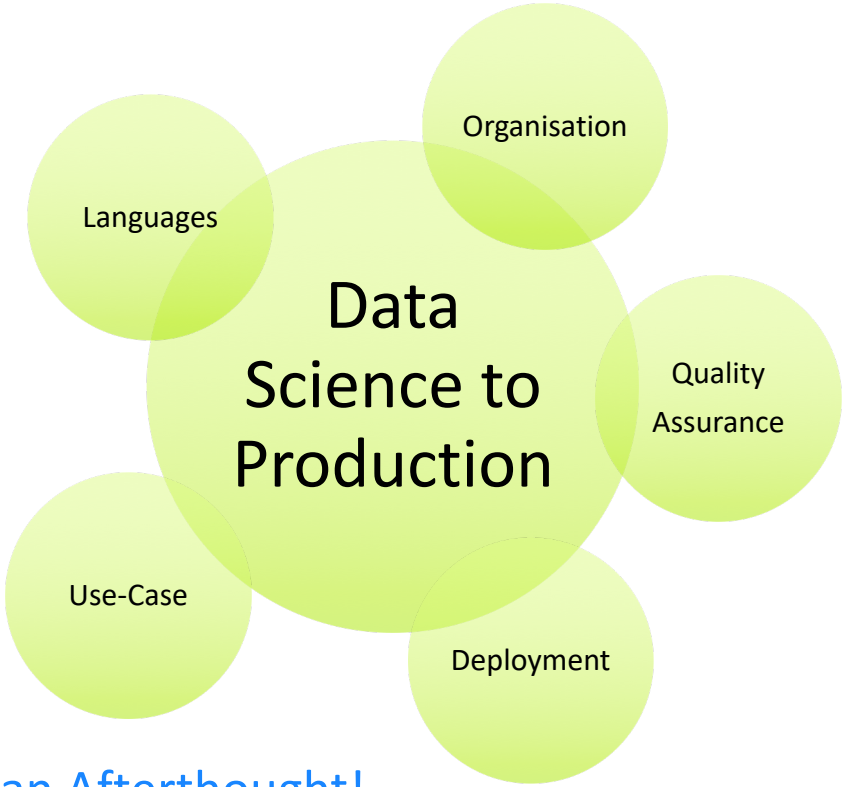



```
$> pip install pyscaffold
$> putup my_project
```

Key Learnings

Data Science to Production

- Dependent on your use-case, no one-size fits all!
- Think early on about QA
- DevOps Culture & team responsibility
- Choose a framework or single language to overcome the Two-Language-Problem
- Embrace processes & automation



Production is NOT an Afterthought!

Thank you!

Florian Wilhelm
Principal Data Scientist

inovex GmbH
Schanzenstraße 6-20
Kupferhütte 1.13
51063 Cologne, Germany
florian.wilhelm@inovex.de