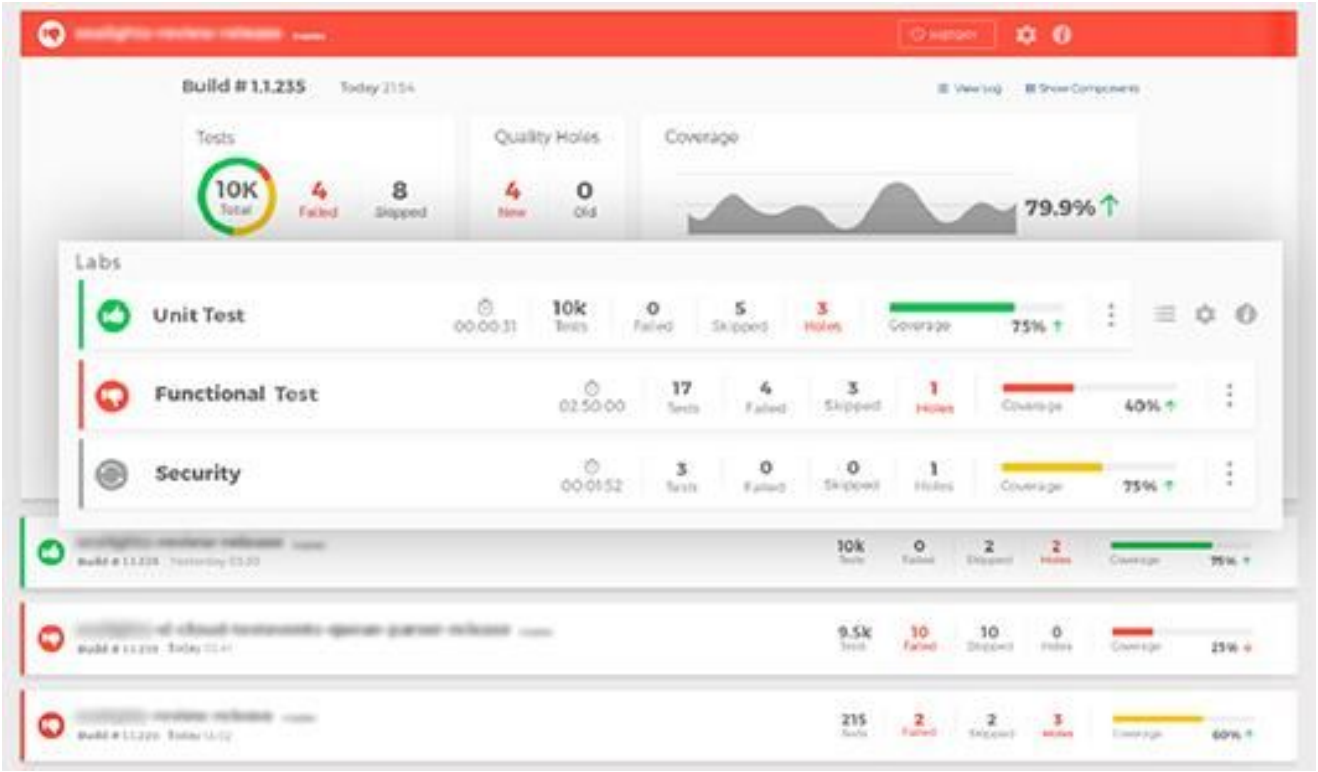
COMS W4156 Advanced Software Engineering (ASE)

December 7, 2021

shared google doc for discussion during class

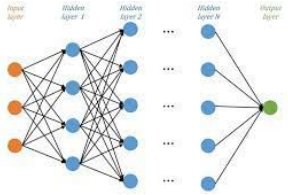
Software Test Results Look Something Like This



Machine Learning Test Results Look Something Like This

Model	[redacted]			
	Accuracy	Precision	Recall	F1
[redacted]	0.497	0.511	0.713	0.595
[redacted]	0.648	0.675	0.619	0.646
[redacted]	0.624	0.631	0.669	0.649
[redacted]	0.723	0.751	0.701	0.725
[redacted]	0.689	0.671	0.738	0.703
[redacted]	0.635	0.589	0.882	0.706
[redacted]	0.689	0.656	0.792	0.717

ML Testing Mismatch



Most real-world software systems are not released until close to 100% of the tests pass and the test suite has close to 100% coverage

Many software systems that include ML components are in real-world use even though the F1 score (from testing ML predictions) is not close to 100%

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

TP = number of true positives

FP = number of false positives

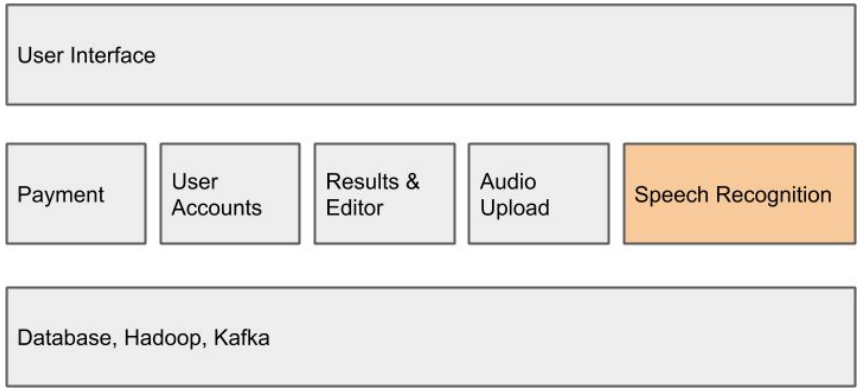
FN = number of false negatives

Regression testing metrics usually treat the ML components as black boxes and do not reflect ML testing metrics like accuracy and F1 score

Software Engineering for Machine Learning (SE4ML)

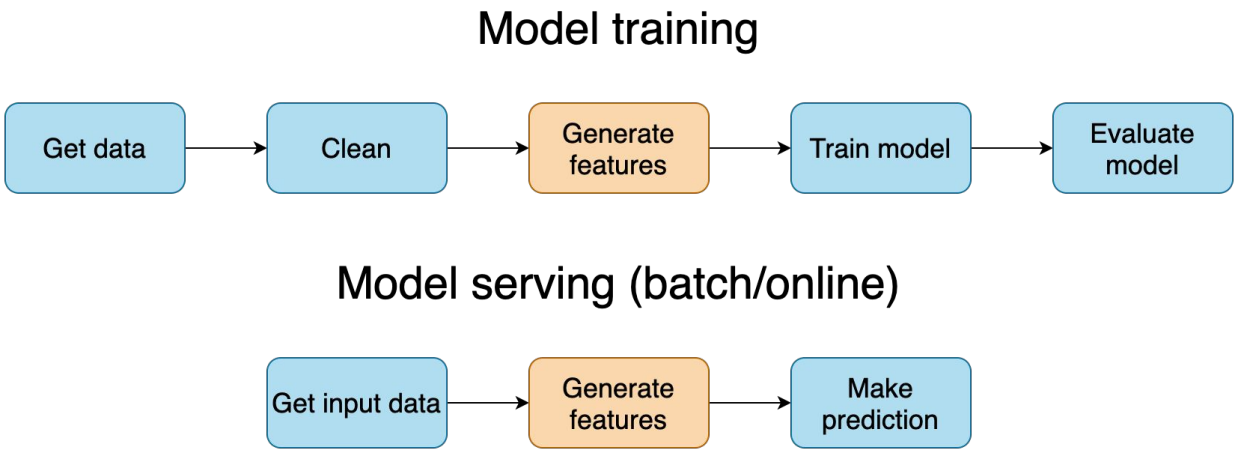
So how do we develop real-world software systems with ML components? For example, a hypothetical online transcription service

The speech recognition component is enabled by an ML model, but the rest is conventional software (e.g., written in Java)



Mismatch between Training Data and Production Data

ML components are highly data dependent: Their performance in production depends on how similar the production data is to the training data, known as "training-serving skew"

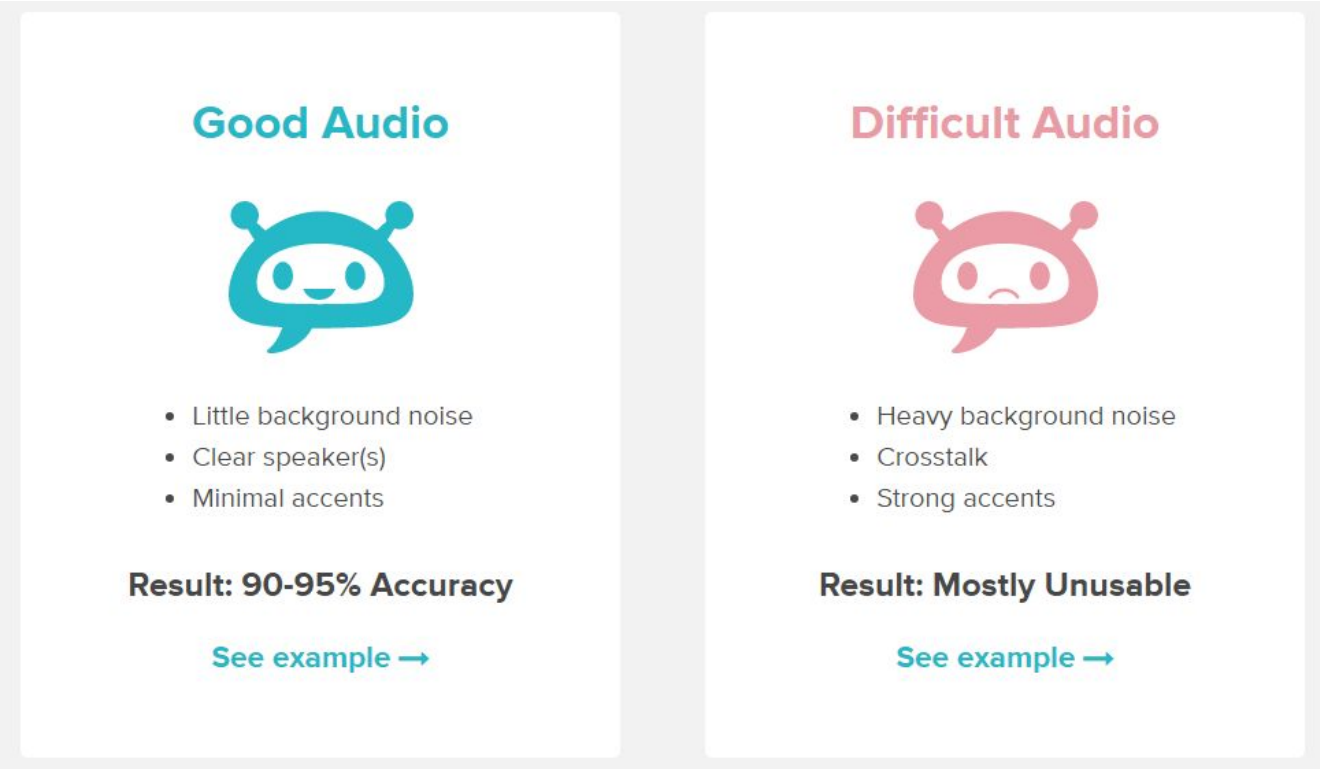


Data Set Mismatch

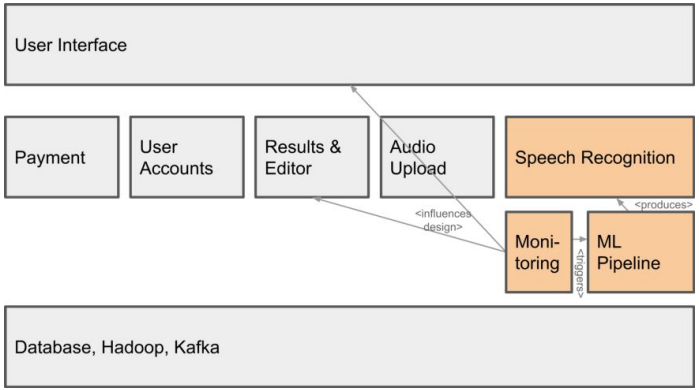
In our hypothetical example, imagine the users are international travelers but the speech recognition model is trained and tested on voice recordings made by US college students (easy to recruit and don't cost much!)

ML-enabled systems must monitor for when model performance degrades and provide enough information to effectively retrain the models based on real users

From a Real Speech Recognition System (temi.com)

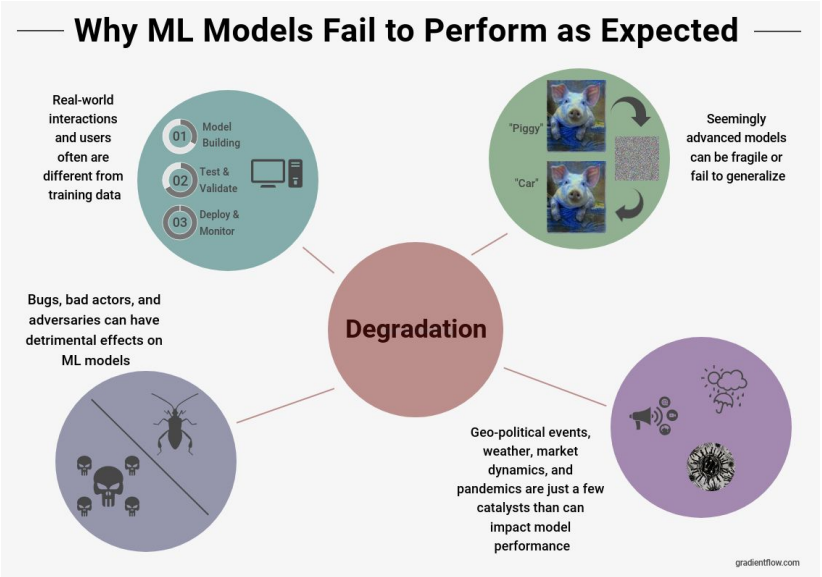


We Need Something Like This



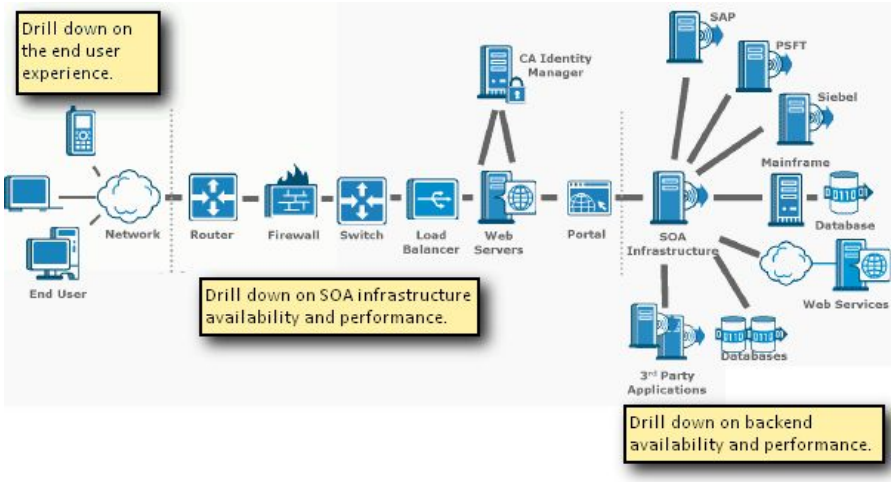
This means adding a monitoring component and an ML pipeline to the production environment

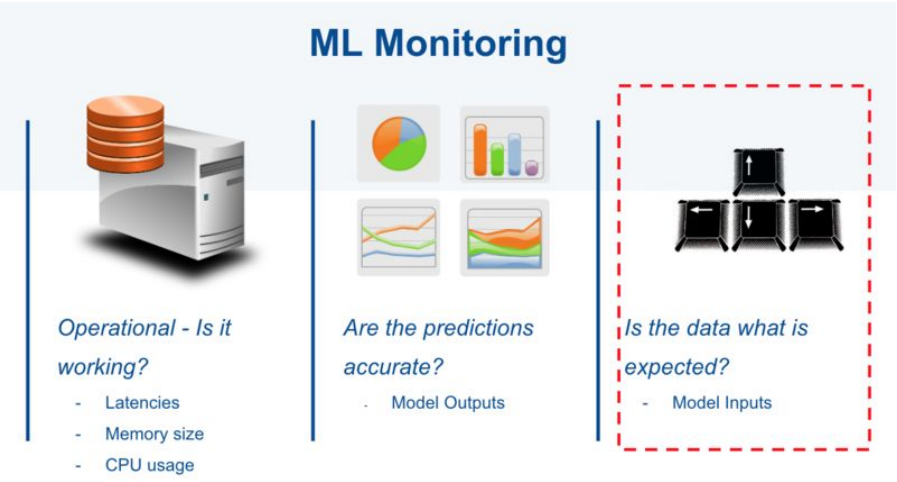
Note monitoring and pipeline are not, themselves, ML components



Monitoring Mismatch

But the “monitoring” services normally used by software engineers measure performance metrics like data throughput, average I/O rate, etc., not ML inputs, outputs and accuracy



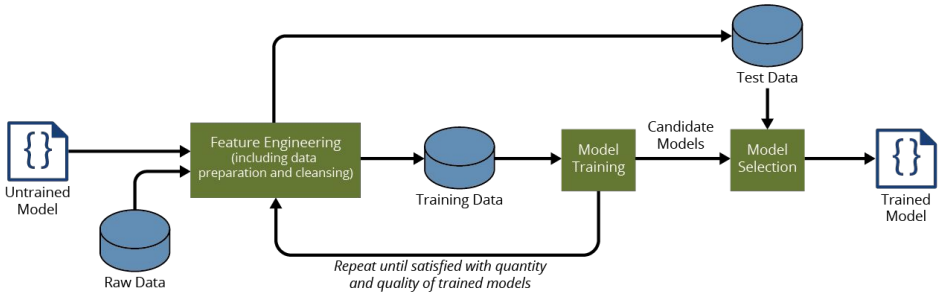


API Mismatch

"Glue code" (adapters) often needed because the ML component expects different inputs and outputs than what is provided by the system into which it is integrated

The API may be simple at a high level, e.g., a REST API with audio input and text output

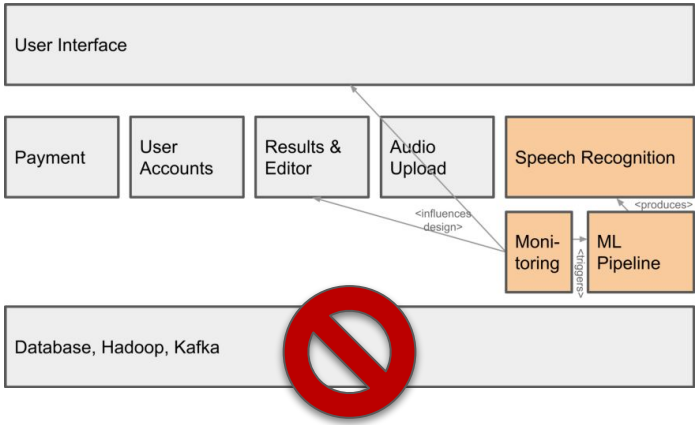
But the ML model may rely on "features", e.g., extracted from metadata, that are not collected or are organized differently by the software system



Computing-Resource Mismatch

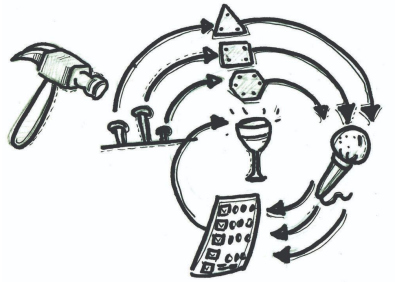
Poor system performance because the computing resources required to execute the model are not available in the production environment

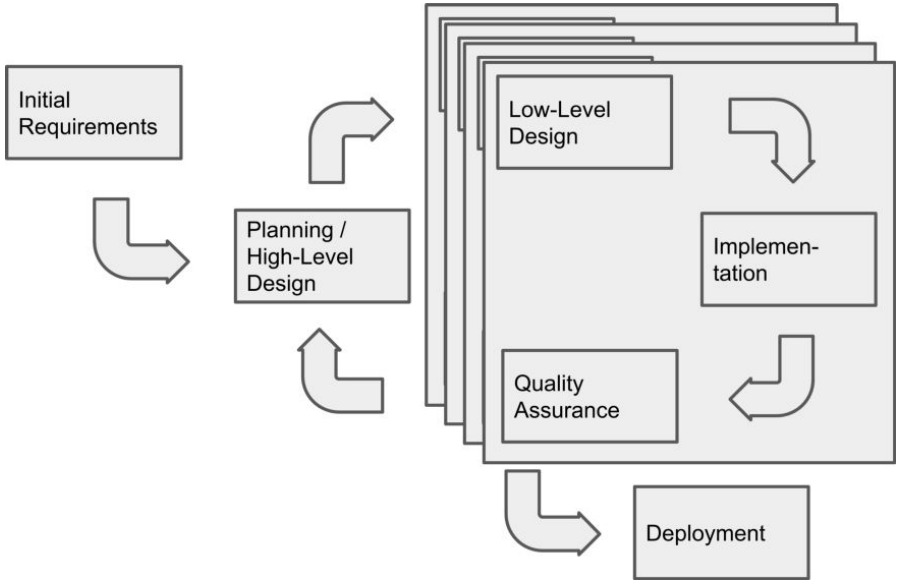
Imagine a mobile version of our hypothetical speech recognition system running on the mobile device rather than sending audio files to a backend server...

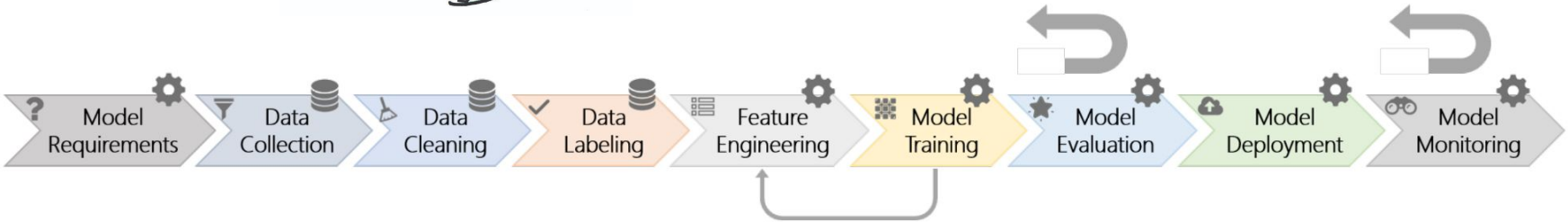


Process Mismatch

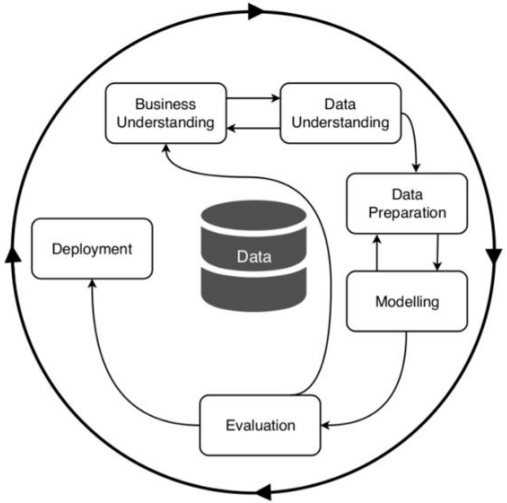
The fundamental problem is software process iteration is different from ML process iteration



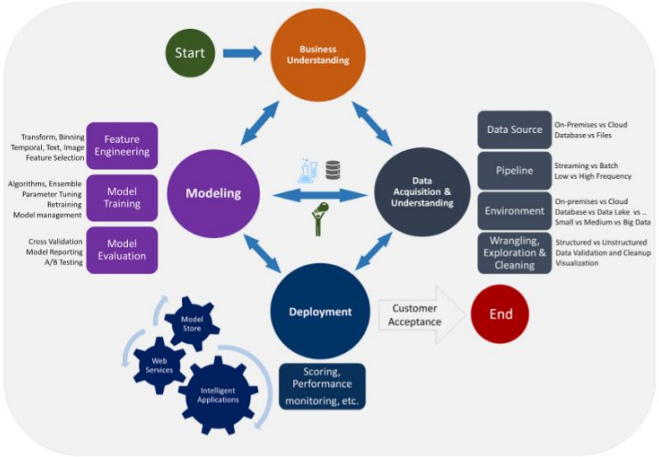




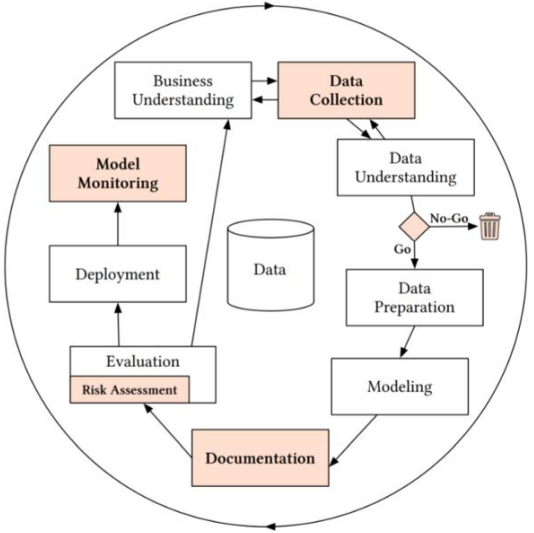
CRISP-DM



Team Data Science Process

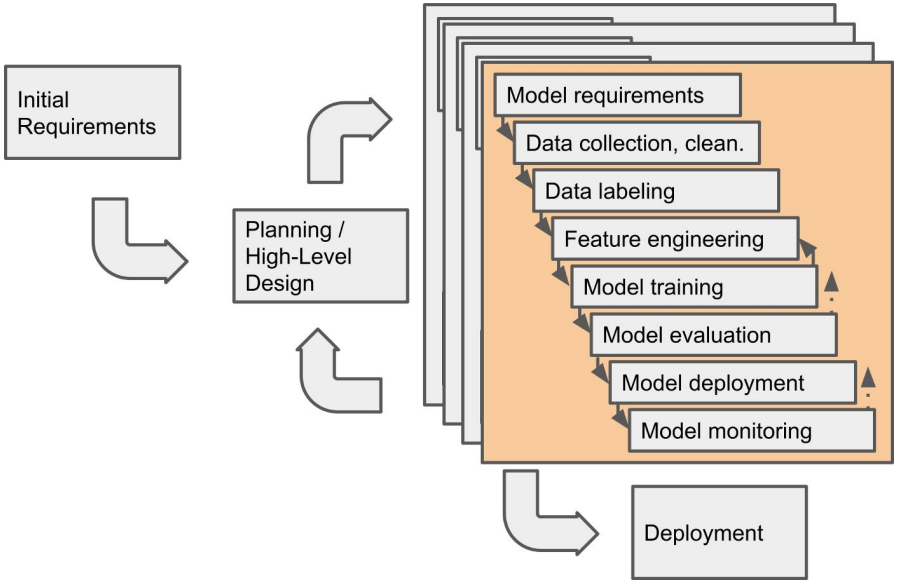


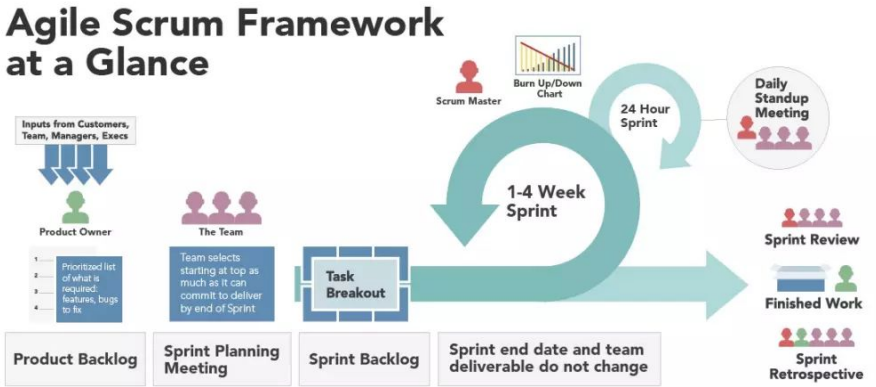
CRISP-DM Refined



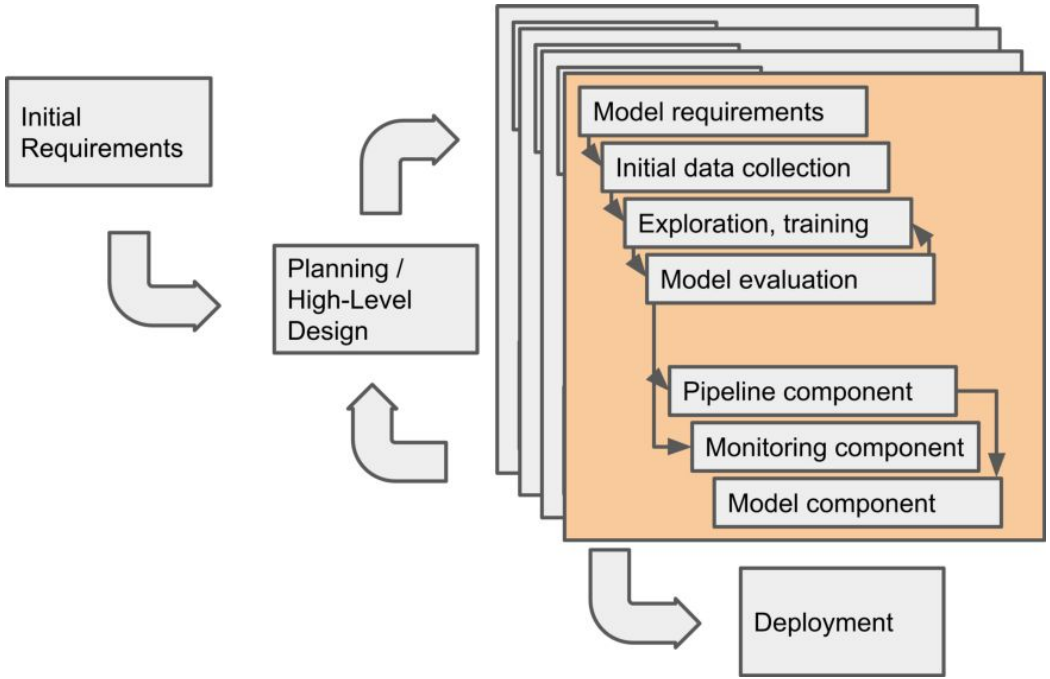
ML workflows do not consider the larger system into with the ML component needs to be integrated with many other non-ML components

ML components are developed with a different process than other components in the software system





And the ML workflow does not produce and deploy the ML model as a single component

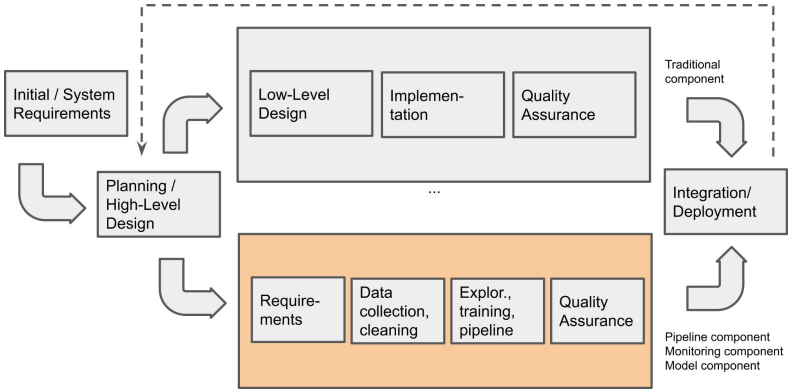


Pipeline component takes data, processes it, and produces and packages an ML model

Monitoring component takes data collected by other parts of the system, produces reports and possibly triggers the pipeline component to rebuild a model

Actual ML model component is produced by the pipeline and integrated into the production system

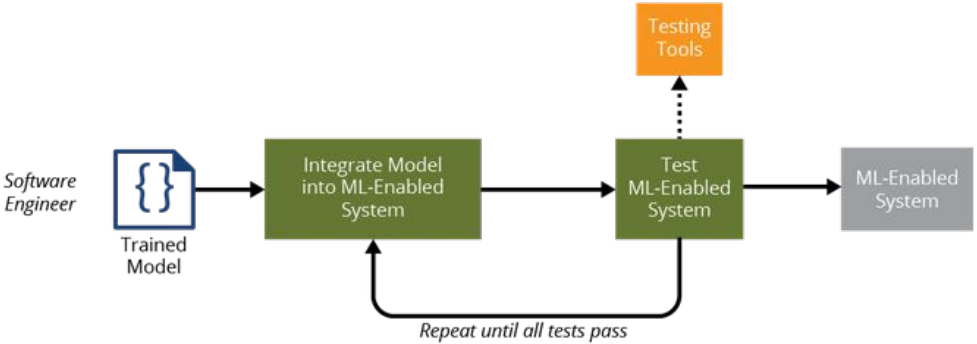
We Need A Process More Like This



Back to Testing Mismatch

Evaluating accuracy of the ML model (component) might be considered analogous to software unit testing: In both cases, we execute the system/model with different inputs and compare the computed outputs with expected outputs

A software test executes the system in a controlled environment with specific inputs, e.g., a function call with specific parameters, and expects specific outputs, e.g. “assertEquals(4, add(2, 2));”



A test suite fails if any single one of the tests does not produce the expected output

There’s a Problem with Unit Testing ML Models

Consider another hypothetical example, predicting house sale prices

Search this file…

	Rooms	CrimeRate	…	PredictedPrice	ActualPrice
2	3	.01	…	230k	250k
3	4	.01	…	530k	498k
4	2	.03	…	210k	211k
5	2	.02	…	219k	210k

```
assertEquals(250000, model.predict([3, .01, ...]));
assertEquals(498000, model.predict([4, .01, ...]));
assertEquals(211000, model.predict([2, .03, ...]));
assertEquals(210000, model.predict([2, .02, ...]));
```

All the assertions fail on the validation data! How do we fix the “bug”?

Performance Testing May Be A Better Analogy

Performance testing evaluates the quality of an implementation with regard to some metric like response time, but usually without specifications and without expecting exact performance behavior - while accepting some nondeterminism and noise

Instead, averaged over multiple executions, possibly with diverse inputs - like evaluating ML accuracy by averaging over validation data

```
@Test(timeout=100)
public void testCompute() {
 expensiveComputation(...);
}
```

Instead of ML model *correctness*, evaluate fit, accuracy, or ... performance

Team Project

Assignment T5: Second Iteration due this Saturday, December 11

Assignment T6: Second Iteration Demo due next week on Wednesday, December 15 (you can do T6 before submitting T5, contact your IA asap for scheduling)

Assignment T7: Demo Day on December 20

Second Individual Assessment will be posted at 12:01am on Thursday, December 16, and due Sunday, December 19 at 11:59pm. There will be another mandatory team participation question - if you do not explain your scoring sufficiently you will get 0 on the assessment