UNIT/ INTEGRATION TESTING

1 THE IMPACT OF EARLY ERROR DETECTION

Data Quality Bugs: where can be caught?

DEVELOPMENT STAGE (Optimal)

Bugs caught during pipeline development (almost like the bug never happened)

Benefits:

- Issue is fixed before production deployment; no trace from production history.
- Protection against dependency changes.
- Enforced quality through CI/CD.
- Cost-effective.

Detection methods:

1 Unit Tests

What: Testing for individual functions (particularly UDFs).

Where: lookup functions in Spark; code integrating with external libraries; individual pipeline components.

2 Integration Tests

What: test how different components work together.

Where: end-to-end pipeline functionality.

Linting [Bonus]

What: enforce coding standards (prevents structural issues more than catching functional bugs)

Why: improve code readability; standardized & consistent coding practices.

PRODUCTION STAGING (Acceptable)

Bugs caught mostly during WAP (Write-Audit-Publish) process

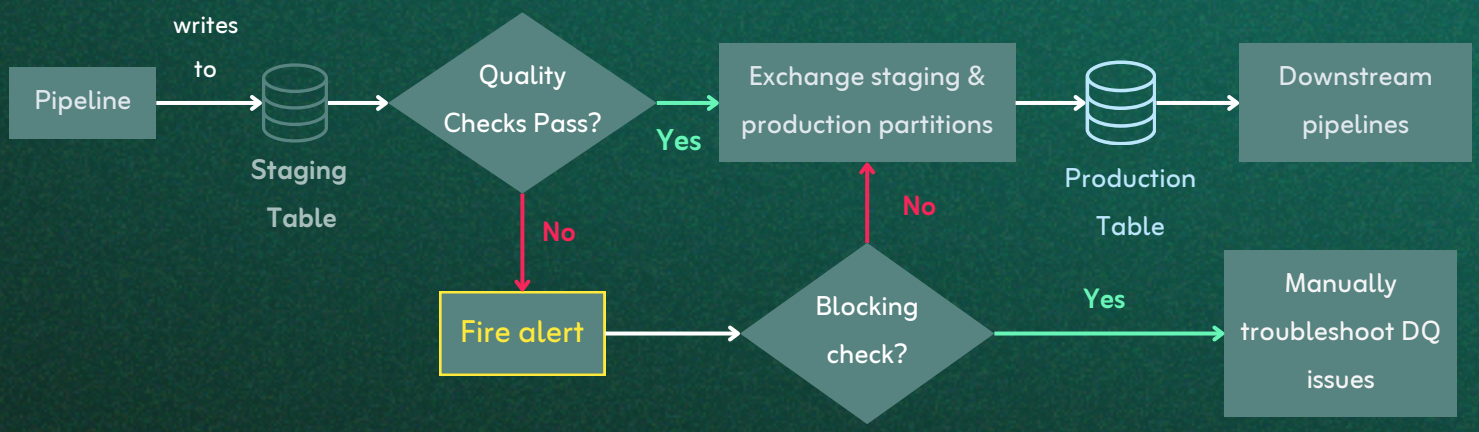
False positives in DQ checks (i) strain Data Engineering teams & (ii) are particularly expensive vs dev stage detection. Goal: minimize these!

Detection methods:

1 WAP

Structured validation process that prevents bad data from reaching production (separates between staging & production).

Process Flow



+ CRITICAL (continued)

Albert Campillo

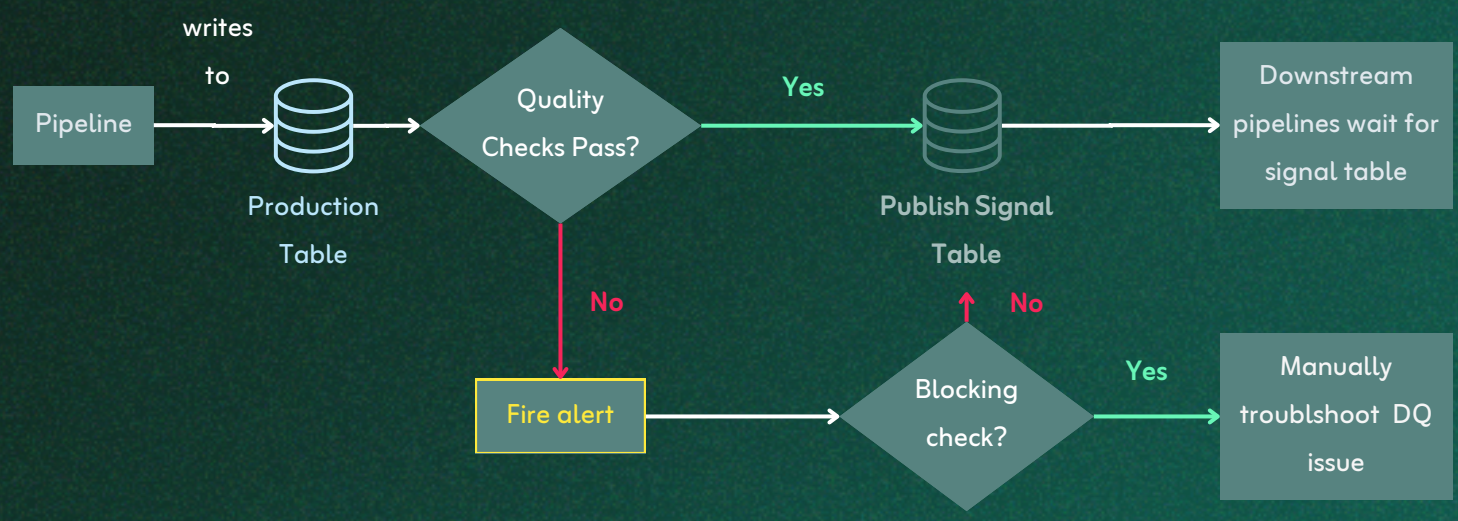
Repost

(continued)

2 Signal table

Successful quality checks on production data are recorded in a separate signal table that downstream pipelines monitor before consuming the data

Process Flow:



PRODUCTION TABLE (Worst case scenario)

Bugs that bypassed DQ checks and are caught in production

Detection methods:

- Anomalies reported by analysts/ shown in dashboards anomalies

Impact

- Destroys trust in data; longer periods w/o detection; most expensive to fix

+ CRITICAL

2 QUALITY STANDARDS

Fact : Software Engineering has higher quality standards than Data Engineering

1. RISK ASSESSMENT & IMPACT

Software Engineering	Data Engineering
Critical impact of failures • Server/app going down impacts revenue/ user experience.	More flexible tolerance • Pipeline failures have less immediate business impact (don't halt operations)
• Requires immediate response to system failures (zero tolerance for downtime or system unavailability)	• More forgiving timeline for issue resolution (acceptable use of historical/stale data as fallback).

Q: Is Facebook login going down a greater business risk than a failure on the data pipeline powering notifications?

(continued)

UNIT/ INTEGRATION TESTING

2. FIELD MATURITY

Software Engineering	Data Engineering
Software engineering maturity • Circa ~50 years of industry evolution	**Data engineering evolution** • Relatively young field (~10-15 years)

• Well-established methodologies:
  ○ Test-driven development (TDD)
  ○ Behavior-driven development (BDD)
  ○ Comprehensive quality assurance frameworks
	• Adopting & adapting SE methodologies
	  ○ Recent adoption of TDD & BDD
	  ○ Evolving quality standards
	  ○ Testing frameworks still maturing

• Agile methodologies well-established • Robust testing culture & expectations	• Growing emphasis of automated testing • Evolution of data-specific quality metrics

3. TALENT BACKGROUND

Software Engineering	Data Engineering
Homogeneous background (generally) • Computer science • Similar educational foundations • Consistent exposure to development practices • Standard career progression paths.	**Diverse background** • Former software engineers, data analysts,... • Various technical backgrounds • Multiple entry paths

3 INCREASING RISK FACTORS IN DATA ENGINEERING

1 MACHINE LEARNING DEPENDENCIES

Data delays effects

- Business impact (less than SE delays, but quantifiable)
- Progressive degradation of model effectiveness

Airbnb's Smart Pricing algorithm sets accommodation pricing for hosts opting in. Data delays (1-2 days) made models less effective & 10-15% revenue loss

2 DQ IMPACT ON EXPERIMENTATION

DQ bugs that persist hide out risks due to not enough checking.

Risks in experimentation (A/B testing)

- Potential for misleading results (false positives, incorrect metric interpretation, Flawed success measurements)
- Impact on business-decision making quality
- Lack of robust validation checks

3 GROWING TRUST & RESPONSIBILITY

Organizational dependency

- Increased reliance on data-driven decision making
- Greater weight of DE responsibilities at senior DE roles
- Higher stakes of DQ issues
- Evolution towards more critical pipelines

UNIT/ INTEGRATION TESTING

FACT: Most organizations miss the mark on data engineering quality

No unit/integration tests or DQ checks (during Zach's time)

FB philosophy: move fast, break things, iterate quickly (DE & SE don't share the same culture)

- Data Engineering (DE) goal: build pipelines as fast as possible (hence DQ checks as a 'nice to have').
- Data Analytics goal: answer questions as quick as possible
- Software Engineering (SE) goal: build something that lasts. at Facebook not the same culture.

Zach's mindset: build high quality data that lasts & answer questions quick, particurlary in a long time scale

Goal: build robust information highways & roads that can be enriched with high quality data and answer questions fast

Build high-quality pipelines
Technical debt management

Answer business questions

\- <----------> + Velocity
\+ <----------> - Sustainability

Trade-off

4 DE FUTURE CAPABILITY IMPROVEMENTS

LATENCY

From Batch to Streaming / Batch to Microbatch pipelines

- Shorten data availability/ processing/ readiness
- More efficient at solving fraud/ bad behavior detection, dynamic pricing

- Higher failure risks (lower latency = more time running)
- Trickier/ more complex DQ check

QUALITY

Test & data quality on the rise

Leverage use of best practices and frameworks:

- Great Expectations
- Deequ (Amazon): computes data quality metrics, verifies constraints defined by dataset producers & publishes datasets to consumers if DQ successful

COMPLETENESS

DE becoming domain knowledge experts through experience.

DE better business understanding by improving communication & getting closer to the business stakeholders.

- Better answers to business problems

USABILITY

DE solved by:

- Build data products (ie. at Netflix, Zach built an API to retrieve data to be read/writen from any URL)
- Better data modeling

UNIT/ INTEGRATION TESTING

5 DE WITH AN SE MINDSET

PRINCIPLES

- Write code meant for humans to read, not for machines to run
- Avoid silent failures
  - Regularly go unnoticed (your worst enemy)
  - ie. In Java, if you come across an exception, better thing to do many times is 'throw' the exception and actually crash the program. This is better than having a silent failure (if it crashes it alerts you at least)
- Promote loud failures
  - Annoying (need troubleshooting) but great
  - Arise from bad data (most of the time) & can show up during testing (CI/CD catches them before deploying to production)
  - Approach: break the pipeline, throw and exception (ie. duplicate row exceptionx)
- DRY code (don't repeat yourself) & YAGNI principles (You aren't gonna need it)
- Design spec documents
- Care about efficiency
  - Data structures
  - Big o notation
  - Understanding JOIN & shuffle