Scaling up in Scope

How to use CiCd and Unit Testing to ensure quality and correctness as your databricks platform grows in size and scope.

Simon Heisterkamp

Databricks User Group Iceland

2025-05-07

1-slide summary

The problem	Literature	Testing	Wrap up
Proof of concept Success leads to growth leads to complexity	Modules and interfaces Unit & Integration Testing CI CD	Strategy Scalability **Testable Code**	Examples Subjective experience Summary

About me

PhD partice physicist

Worked at CERN

Datasets counted in peta bytes (1.000.000 Tb)

Co-author of Higgs particle discovery

8 years experience in defense and energy

Systems Engineering

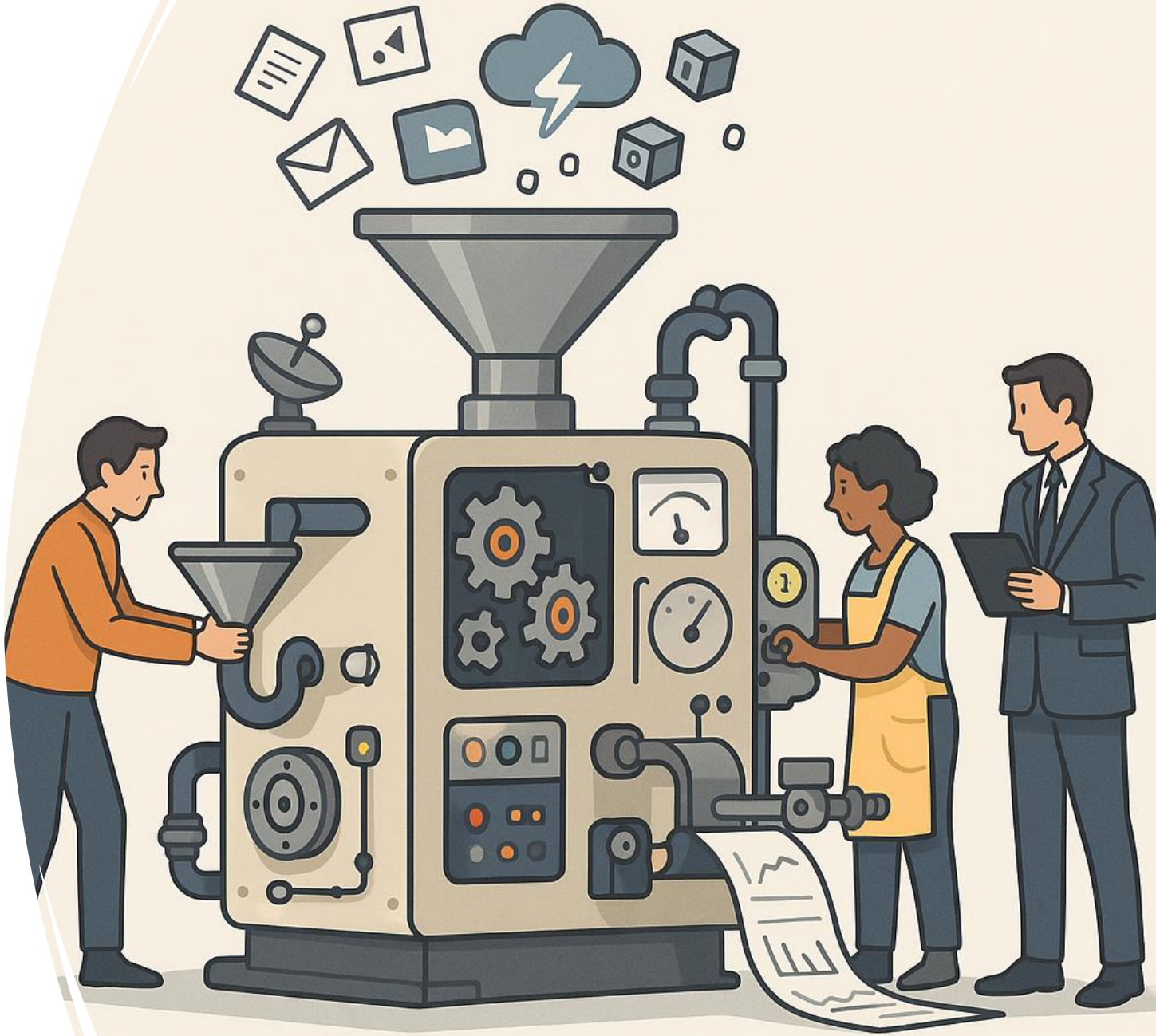
Testing and Quality Control

4 years experience with Databricks

Freelance Developer since 2025

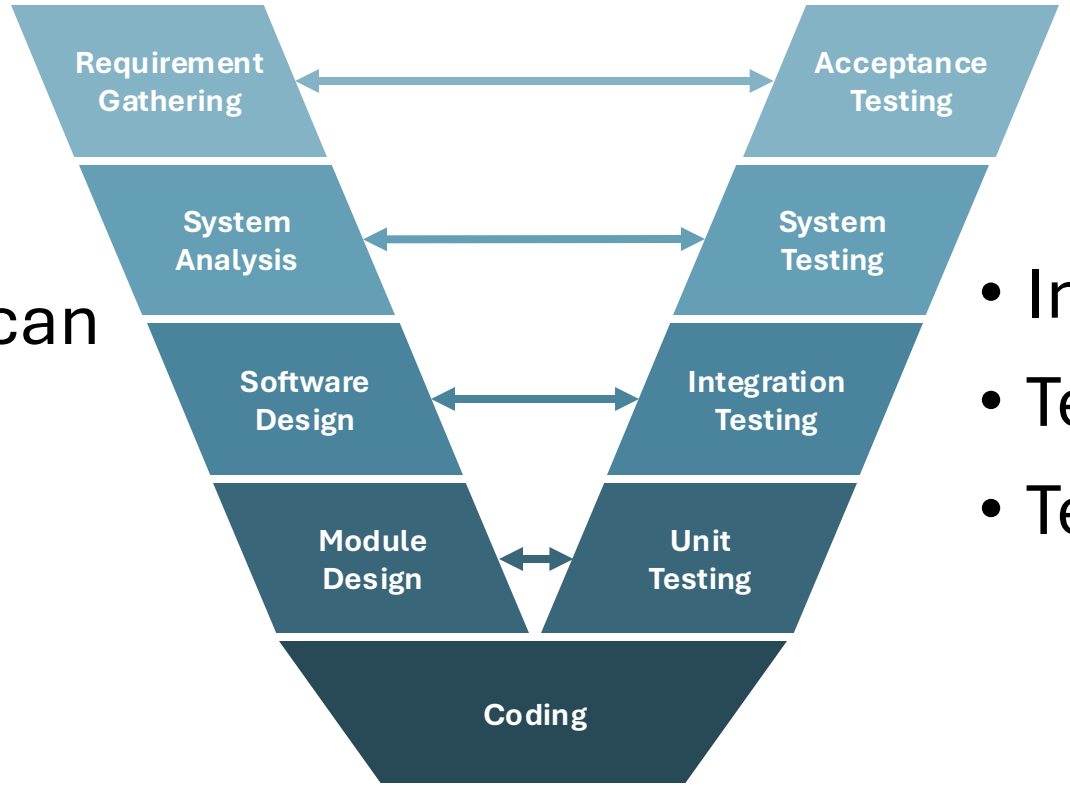
Starting point

- The proof-of-concept works
- Build on success – add functions
- Reuse components
  - tables & code
- Ch-ch-ch-changes…
- How to maintain quality?



Systems Engineering

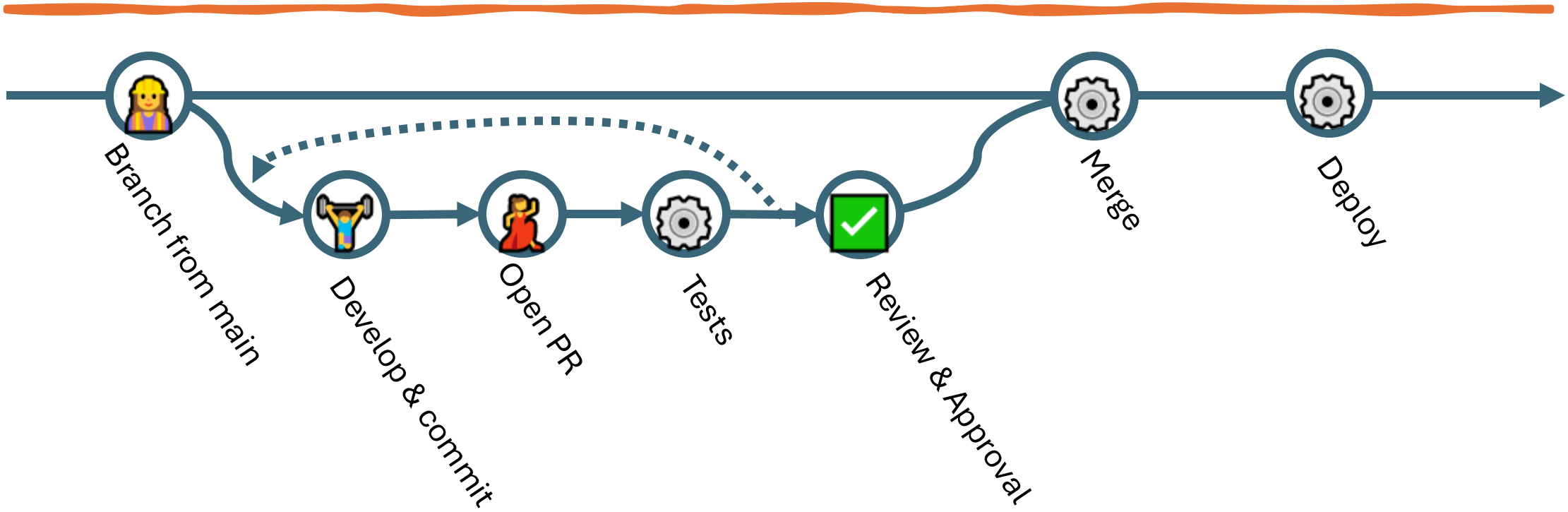
- **Any** complexity can be managed
- Subdivide
- Assign functions
- Simple units



- Integrate parts upwards
- Test every level
- Test every integration

- In practice: iterate!

The PR flow & CICD

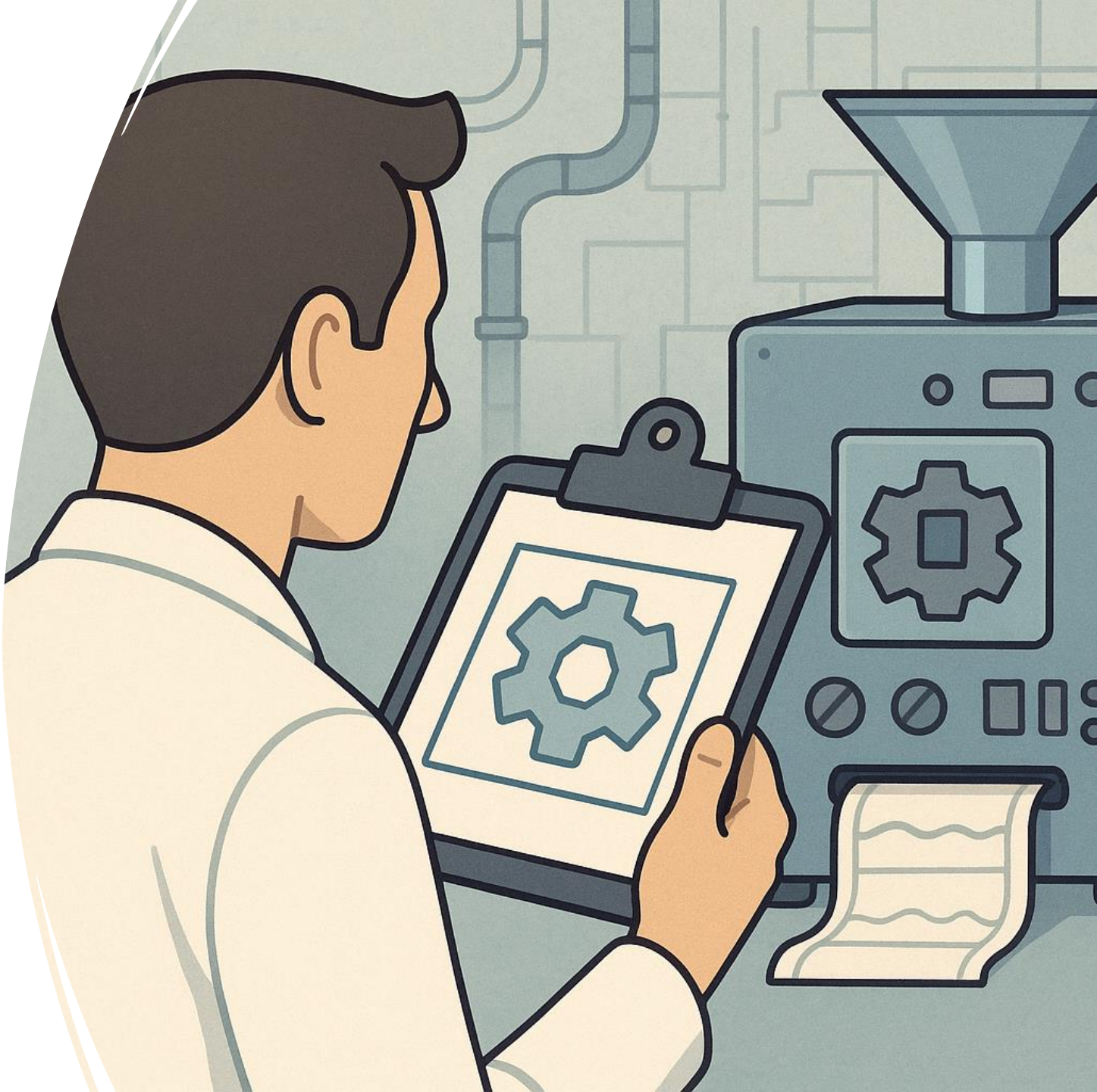


- Test and deploy pipelines are custom code
- Continuous Integration – test often
- Continuous deployment – deploy often

So far – so theoretical…

In repo: ETL code and table schemas

1. Challenge: Does the code run with described schemas?
2. Challenge: Does the code run on the production context?
3. Challenge: What to do with deployed schemas on repo changes?



Strategy

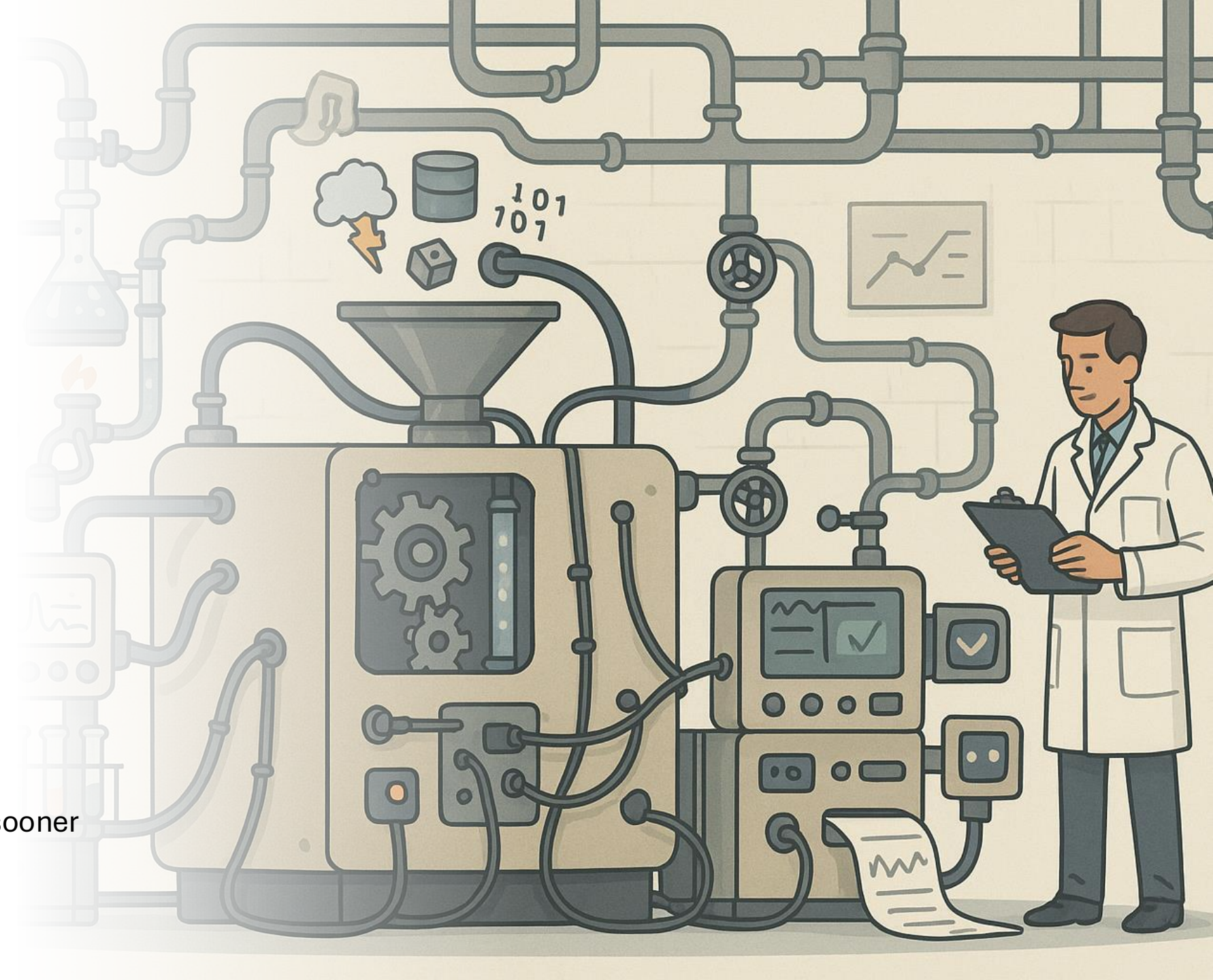
Manual Testing

- Low setup
- Conceptually simple
- Slow per test

Automated Testing

- Larger Setup
- Maintenance
- Fragile Tests
- Test code is also Code

- Test often → find problems sooner
- Don’t automate too late!



Strategy

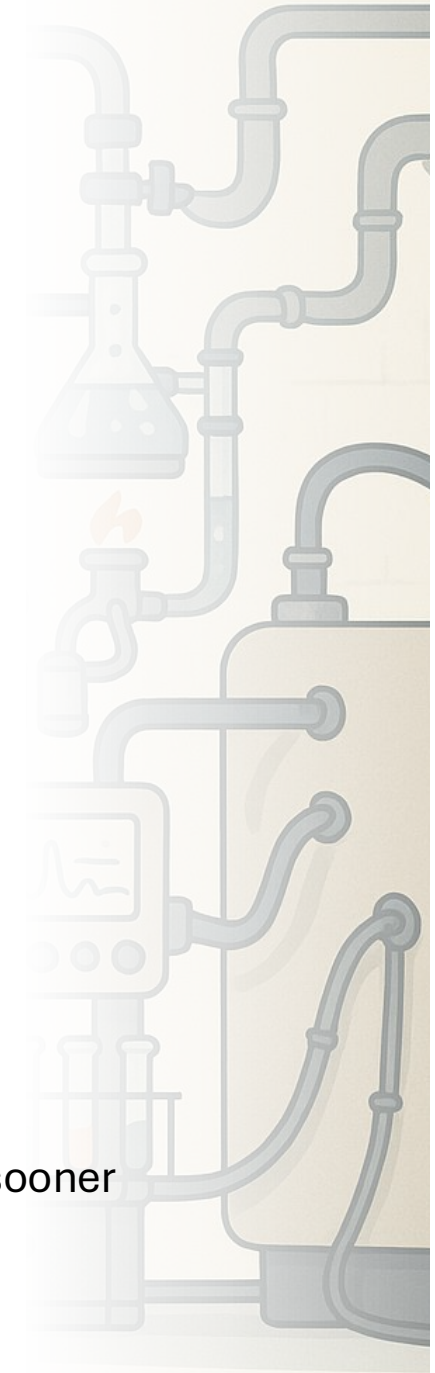
Manual Testing

- Low setup
- Conceptually simple
- Slow per test

Automated Testing

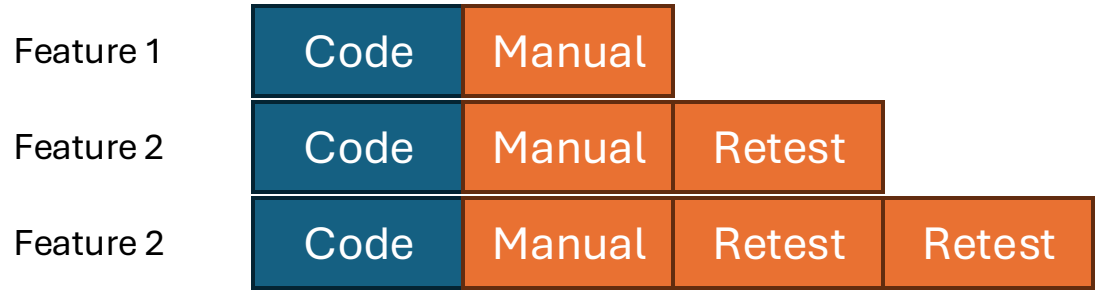
- Larger Setup
- Maintenance
- Fragile Tests
- Test code is also Code

- Test often → find problems sooner
- Don’t automate too late!

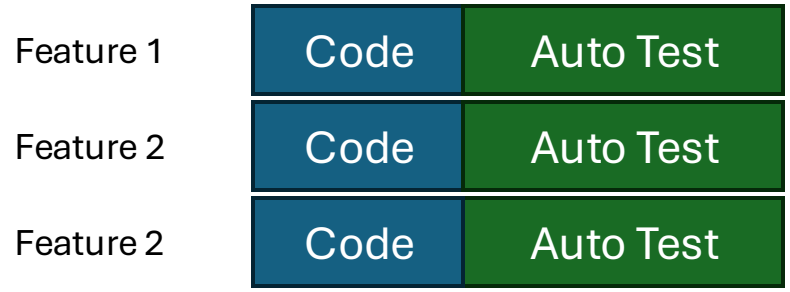


Scalability

Manual Testing Effort

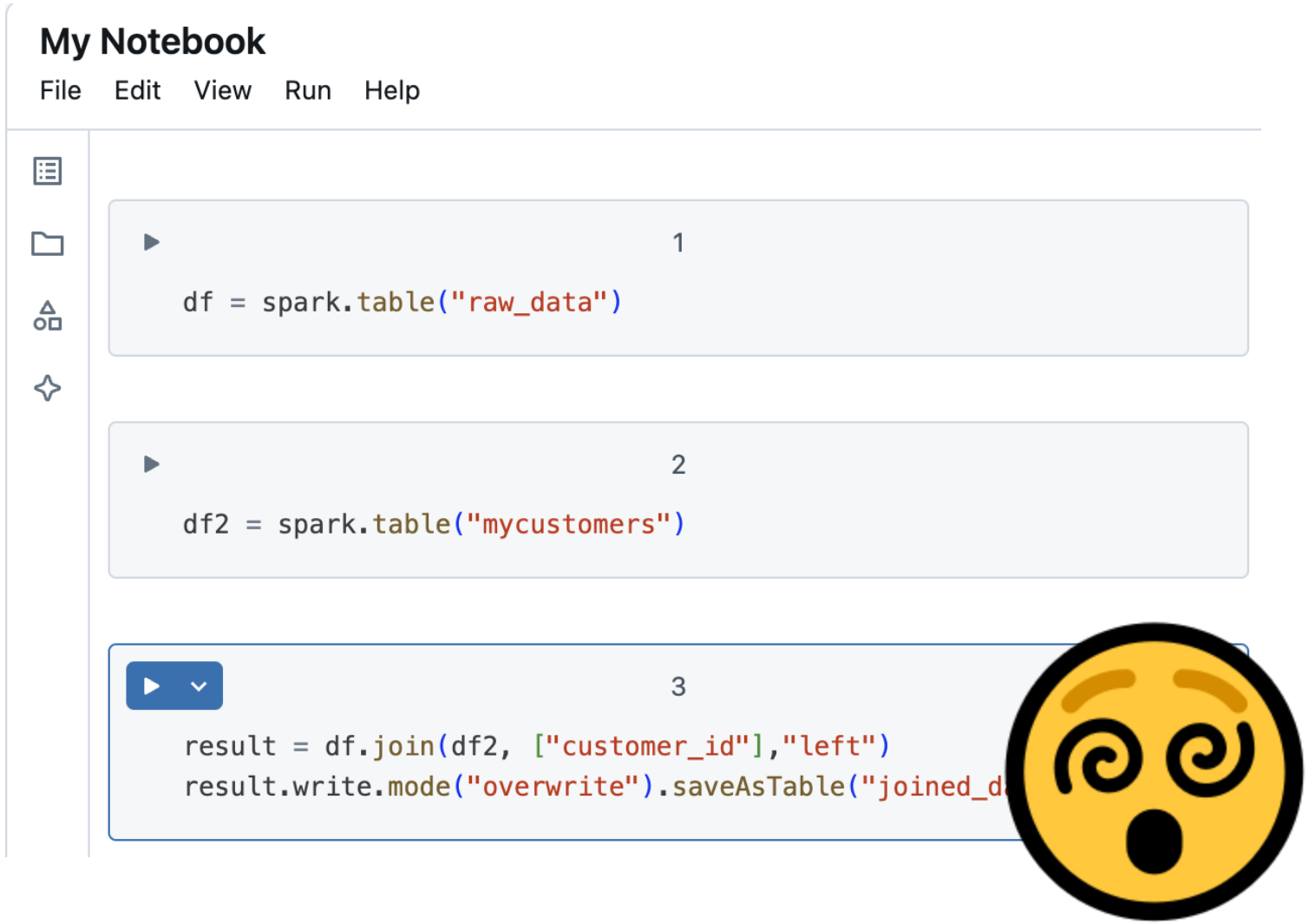


Automated Testing Effort



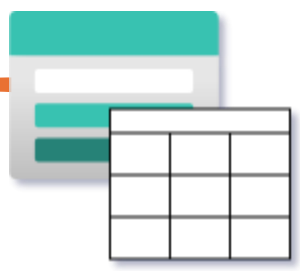
How to Automate?

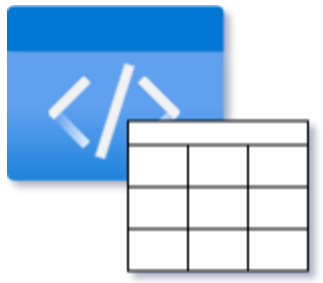
- Context
  - Local spark
  - Databricks Connect
  - **job cluster**
- Framework
  - Pytest
  - Unittest
- Modules
  - Notebook reuse
  - **Object oriented python library**
- Instrumentation
  - Mocking
  - **Dependency injection**

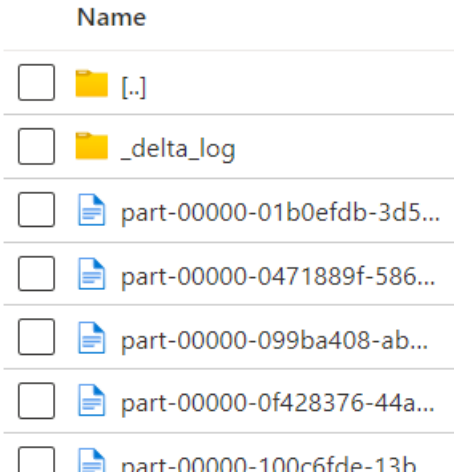


Example: Table Abstraction

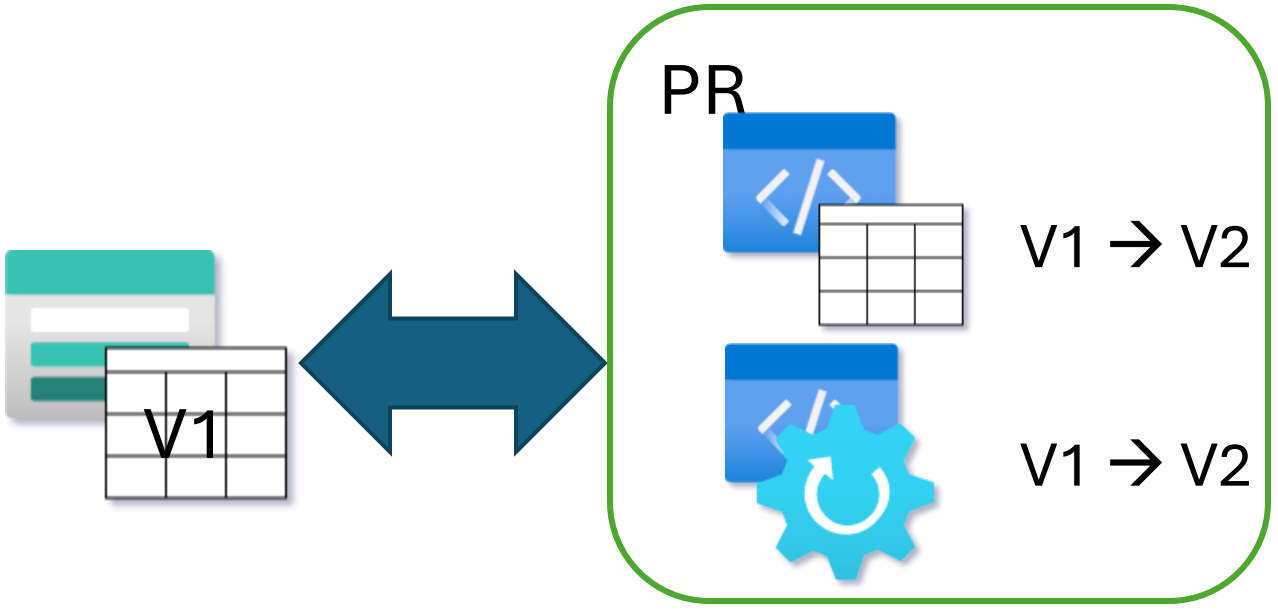
- Every table exists in two places
  - As data – with schema and contents
  - As a concept in code
- “desired” vs “actual” state








```
CREATE TABLE IF NOT EXISTS my_db.detai
(
  my_int INT,
  my_string STRING,
  my_time TIMETAMP
)
USING DELTA
```




```
df = spark.[redacted]("raw_data")
```

Example: Table Abstraction - 2

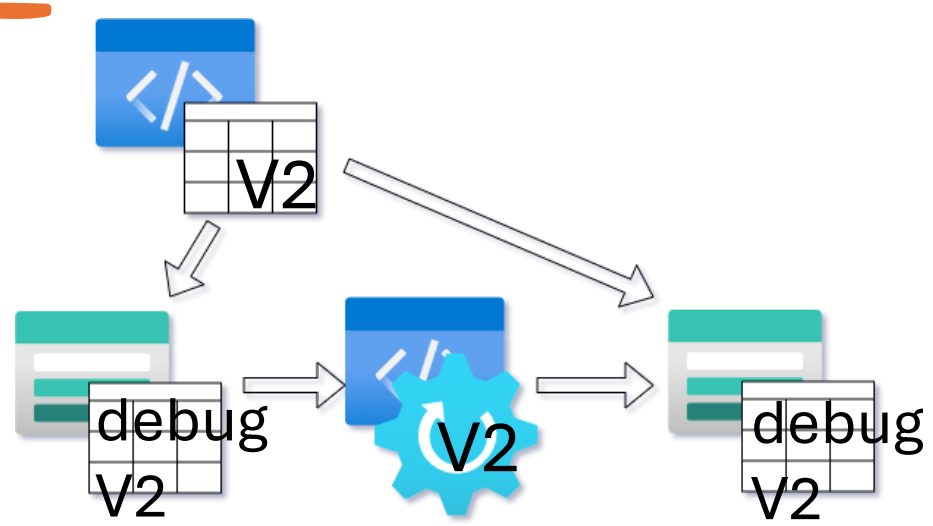
- Standard table creation
- Debug tables for testing
- New abstraction layer
- Tooling support

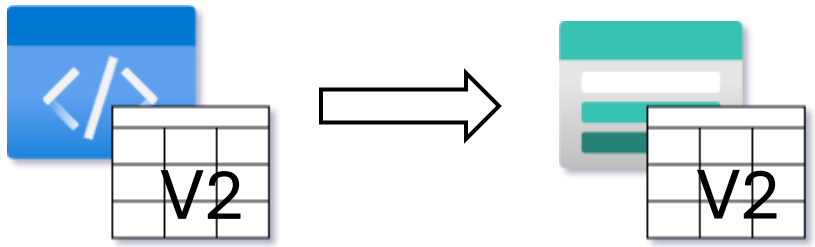
```python
from spetlr import Configurator
from spetlr.delta import DeltaHandle

from demo import sql

# configure
c = Configurator()
c.add_sql_resource_path(sql)
# read production tables
df = DeltaHandle.from_tc("MyDataTable").read()
# select debug tables
c.set_debug()
df2 = DeltaHandle.from_tc("MyDataTable").read()
```

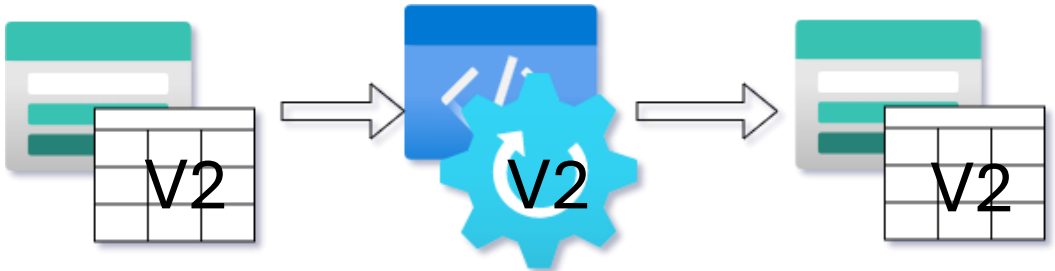
```sql
-- SPETLR.CONFIGURATOR key: MyDataTable
CREATE TABLE IF NOT EXISTS raw_data{ID}
(
  item_id STRING,
  timePoint TIMESTAMP,
  powerWatt Decimal(12,2),
  chargingStatus INT,
  powerGridZone STRING,
  cost DECIMAL(12,2),
  idTokenCustomerCrmId STRING
  P_startConsumptionDate DATE
)
```

Example: Table Abstraction - 3

- Testing:
  - Create debug tables
  - Check ETL can run
- Deploy Tables – compare schema:
  - Match? No change.
  - Mismatch?
    - Recreate
    - Append
    - Error
- Deploy ETL







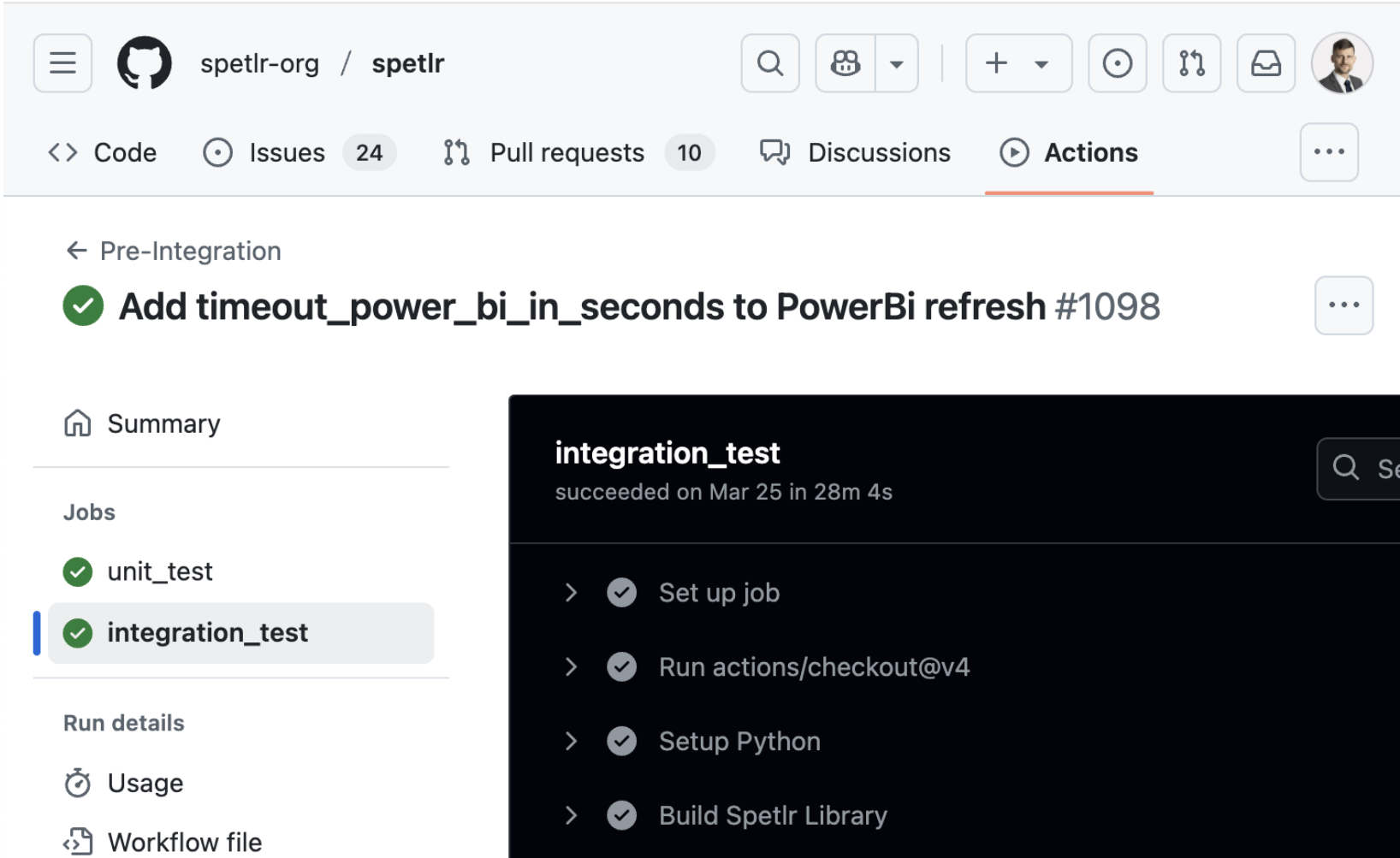
Real world examples

Spetlr

- ETL: 12528 loc
- Tests: 15192 loc
- Infrastructure and pipelines: 2192 loc

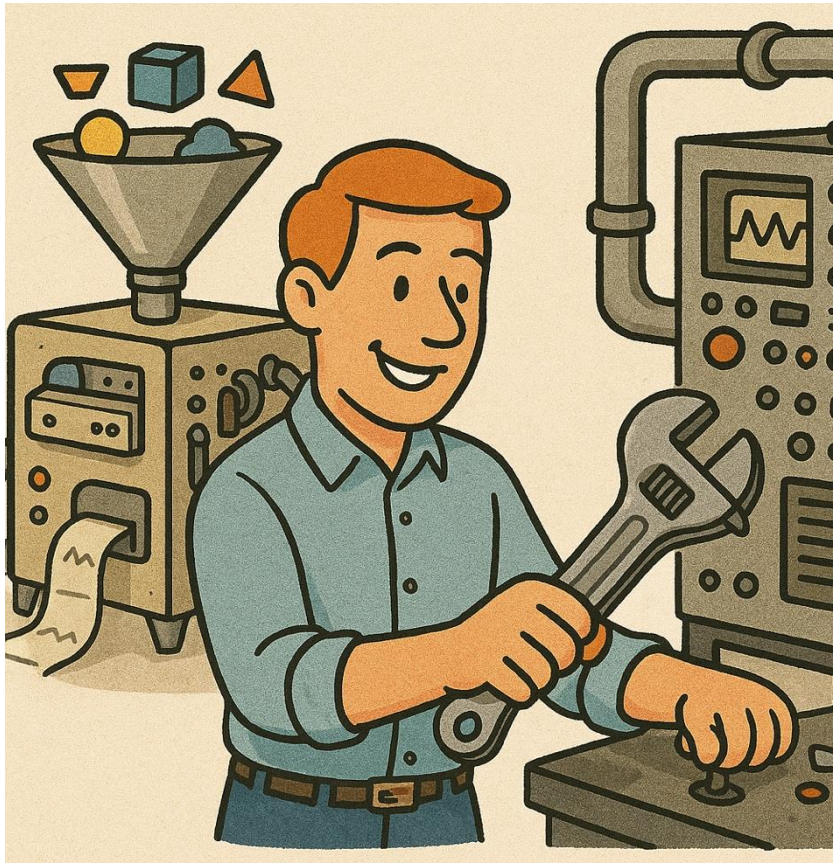
Clever A/S

- ETL: 74241 loc
- Tests: 29853 loc
- Infrastructure and pipelines: 6578 loc



Psychological Aspect

- Engineers enjoy creating elegant algorithms
- Engineers generally don’t enjoy manual repetitive task
- Automating tests draws on **core strengths** of your team



Summary

- Infrastructure as Code (terraform)
- Non-production environments (dev-staging-prod)
- Catalogs – Databases – Tables (schema & properties) all as code
- Modularize, then unit-test modules
- **Make code testable** & test integration of modules
- ETL in python library – easier to test than notebooks
- Test in job cluster – test runtime and all libraries
- PR flow – all changes tested before merge
- Deploy tables changes

”Test as you fly” – NASA

Questions?

Source:

spetlr.com

Get in touch:

linkedin.com/in/simon-heisterkamp