DevOps for Petabyte-scale relational databases

DevOps Pro Moscow 2018

Vladimir Filimonov, Teradata

November 15, 2018

Who are you?

Vladimir Filimonov

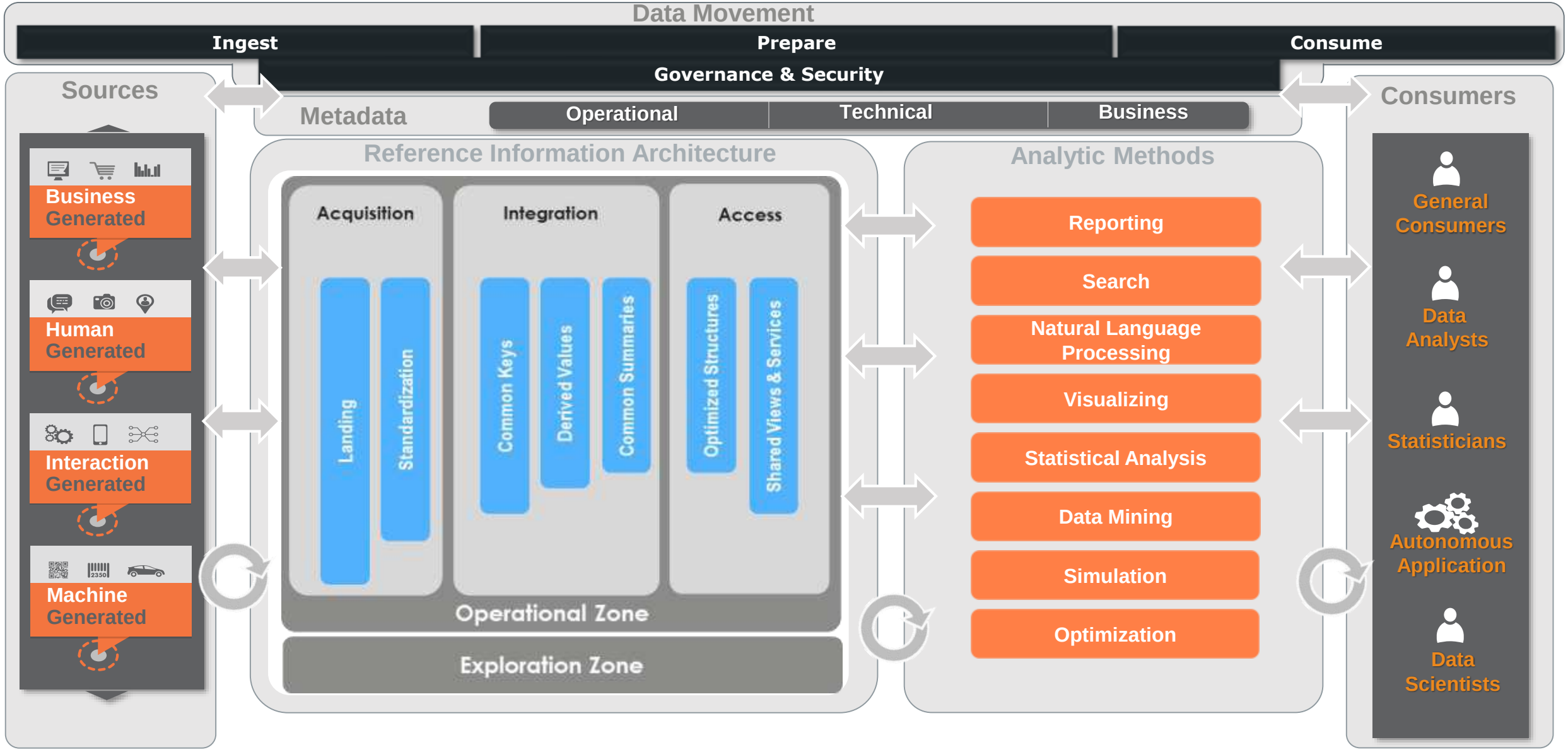
- First computer at age of 4
- Enterprise software development experience ~12 years
- Last 8 years focused on massive data analytics
- Solutions Architect at Teradata
- Live in Moscow
- Like squash and snowboarding

Context

Enterprise analytical ecosystems that include RDBMS

- DBMS: Teradata + others
- Environments: Mega-major enterprise companies
- Massive data volumes: 100s of Terabytes to Petabytes
- Normalized data models
- You can expect schema modification every day
- 10s to 100s developers concurrently creating apps on tops of data platforms
- Complex data flows (ETL/ELT workflows)
- Thousands of users / data consumers

Analytic Ecosystem



teradata.

Typical stages

Many roles,
many manual steps,
many errors…

The journey to
PROD is so long.

Propagation of any
simple change is very
expensive!

We did not get
what we expected!

Maintenance is
almost impossible
after a couple of
years.

- Business
- Modeler
- Analysts
- Developer
- Tester
- Operations

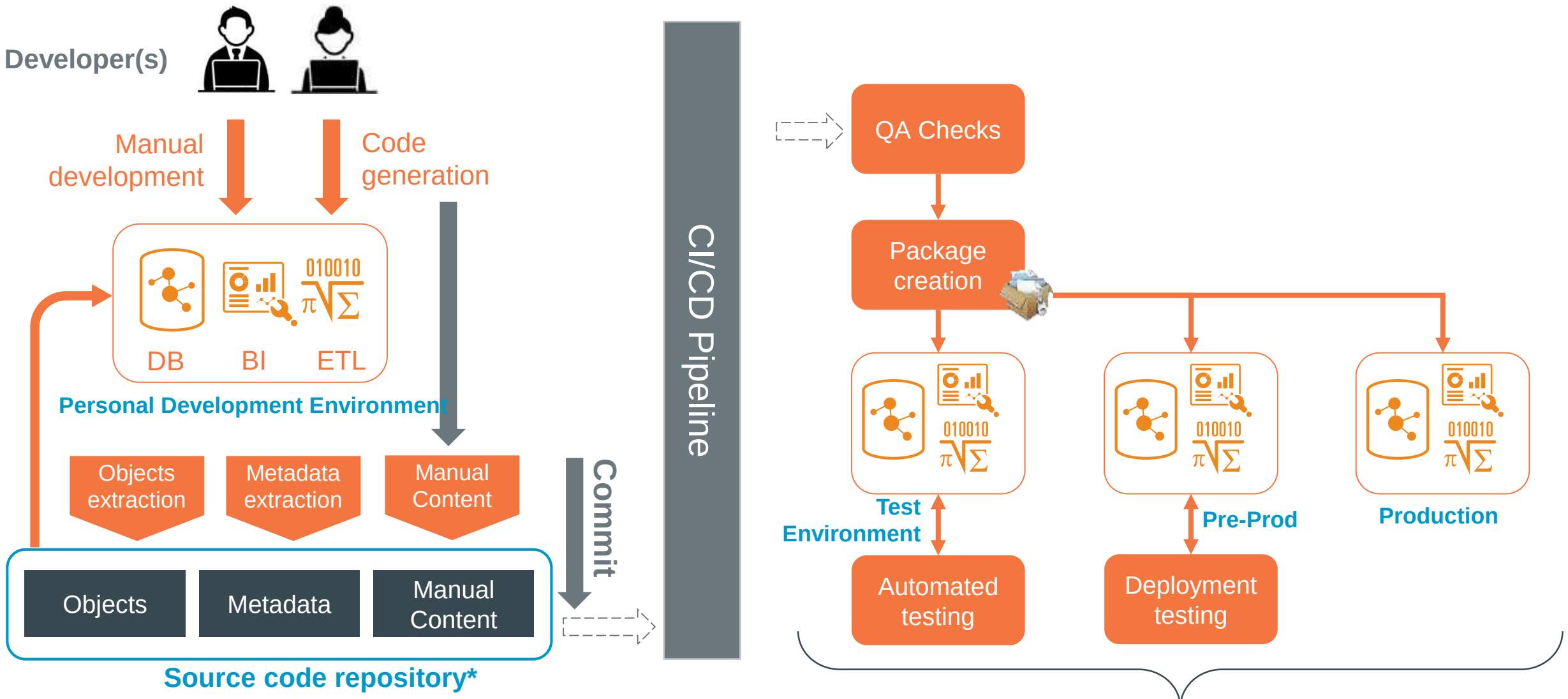
teradata.

What can be delivered as part of release

In any combinations:

- Schema modifications (SQL/DDL)
- Data fixes (SQL/DML)
- Data migration scripts (SQL/DML)
- Database objects changes (beside schema) – like transformation procedures, views, indexes, statistics etc. (SQL/DDL DCL)
- Data load jobs (ETL/ELT, depends on tool)
- Standardized (not self-service) data analysis entities (ex: BI reports, depends on tool)

What does a successful DevOps process looks like?



* only application code part is shown

teradata.

“

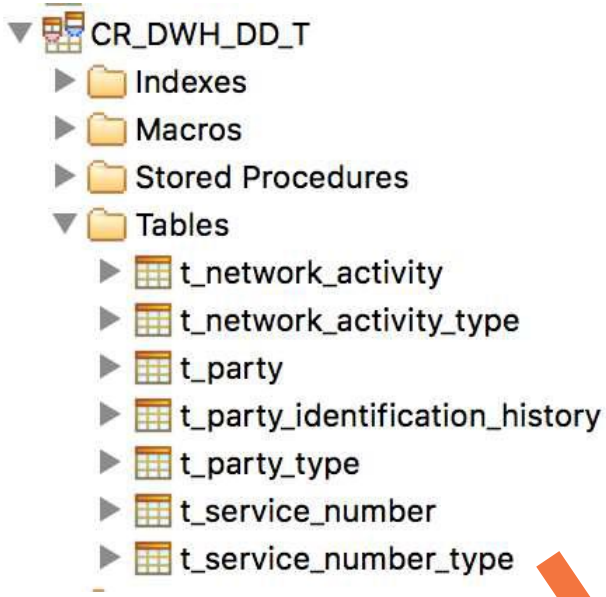
Teams that do well at continuous delivery store database changes as scripts in version control and manage these changes in the same way as production application changes

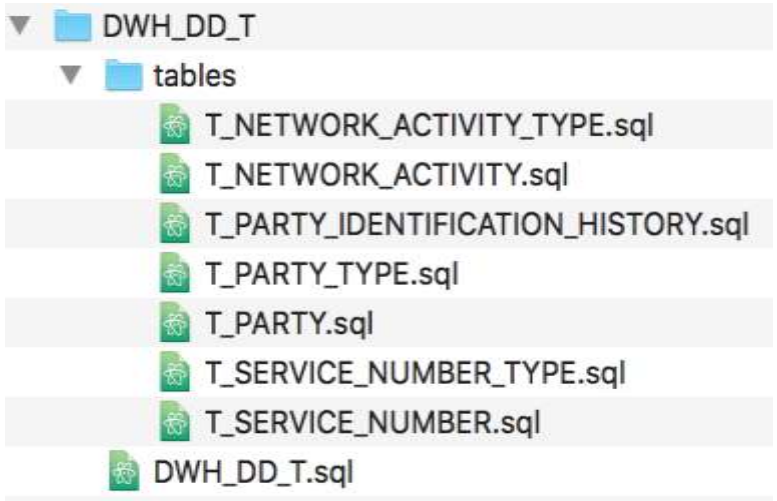
– DORA, State of DevOps report, 2018

Source version control

git

- Everything is code
- Monorepo
- Each objects in separate file
- Clear directories structure
- Clear naming convention of files
- Clear branching strategy (GitFlow is good for the start)
- Early detection of potential conflicts



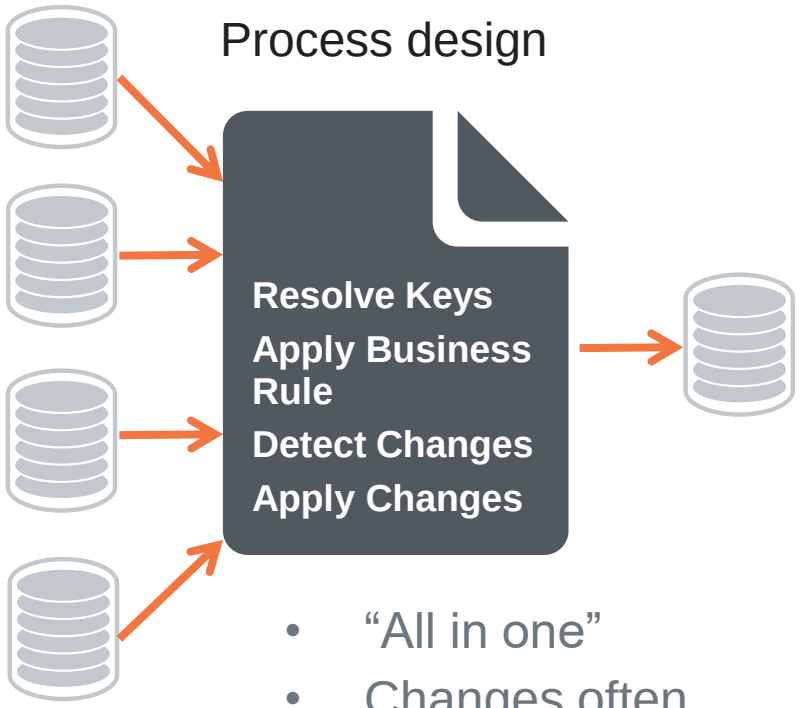


teradata.

Modular Design – controlling scope

Monolithic

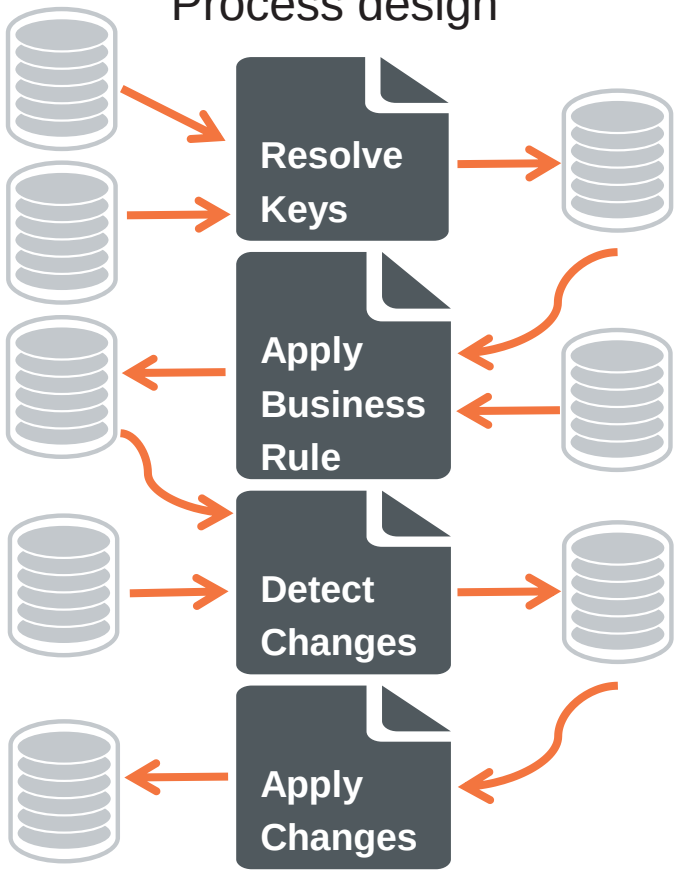
Process design



- "All in one"
- Changes often
- Multiple business rules
- Longer to test
- Harder to get good test coverage

Modular

Process design



- Atomic processing
- Single business rule
- Change rate reduced
- Quick to test
- Easier to get good test coverage
- Easier to automate

teradata.

Code build

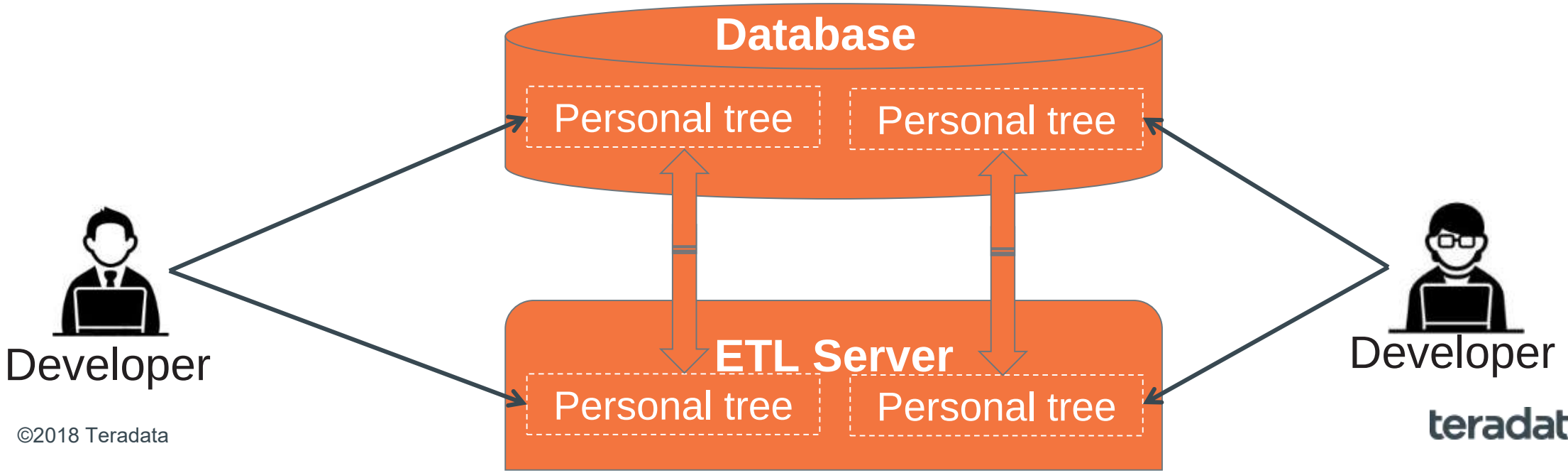
- Build environments from the code
  - Logically separated environments on same infrastructure
  - Totally separate environments with help of VMs
- Respect dependencies
- Partial builds
- Code tokenization
- For all tools in stack

Logically separated environments

For example: On-prem environments

Separate subtrees

- Database – users\databases\schema prefix
- ETL server – separate root folder for structure
- HDFS – separate root folder for structure



teradata.

Functional testing in data systems

- Tests must be the code, under version control
- Testing framework
- Test data management
- Common types of tests
  - Test state of data after data transformation execution
  - Collect typical KPIs of data (count, nulls, RI, etc)
  - Coding standards check

Three steps of component test

Test Pages: 1 right, 0 wrong, 0 ignored, 0 exceptions **Assertions:** 2 right, 0 wrong, 0 ignored, 0 exceptions (13.715 seconds)

Test System: fit:fit.FitServer

✚ *Included page: .FrontPage.SetUp (edit)* Expand Collapse

Create table for error data

Execute Ddl	CREATE MULTISET TABLE FV4_ADWH_DQ_T.TST_SpCreateErrorTable (FAKEFIELD VARCHAR(100))

Insert	FV4_ADWH_DQ_T.TST_SpCreateErrorTable
FAKEFIELD	
Anyrow	

Call SP, check return value

Execute Procedure	FV4_adwh_dq_p.SP_FINALIZE_ERROR_TABLE
IN_TABLE_NAME	OUT_ERRORS_COUNT?
TST_SpCreateErrorTable	1

Check that table still exists

Query Stats	
query	is empty?
SELECT tablename from dbc.tablesv where databasename='FV4_ADWH_DQ_T' and tablename='TST_SpCreateErrorTable'	false

Clean up the test data

Execute Ddl	DROP TABLE FV4_ADWH_DQ_T.TST_SpCreateErrorTable

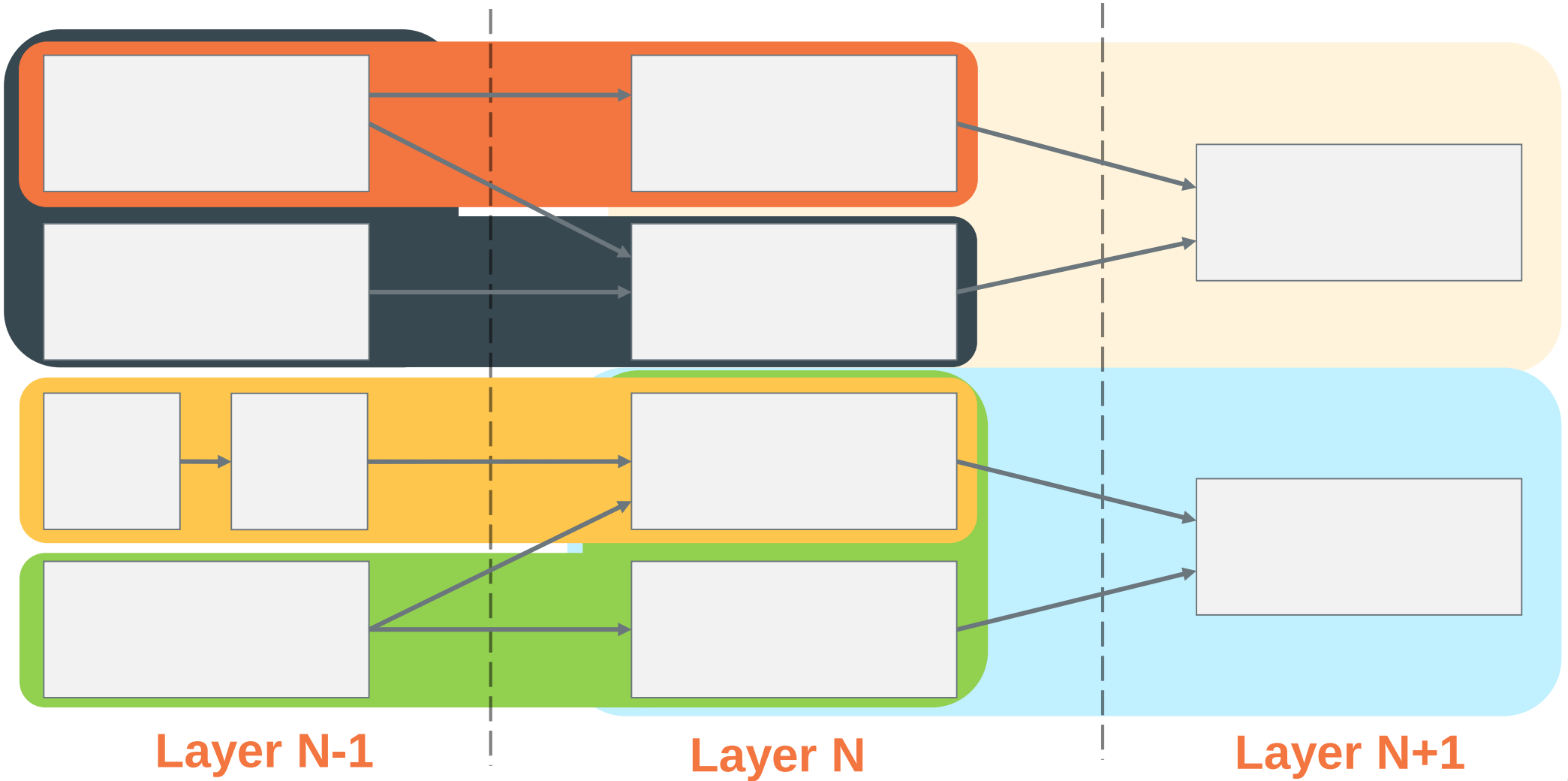
1. Test data preparation

2. Test execution and validation

3. Clean test data

teradata.

Optimize testing scope



Incremental package for database changes

- We need to be incremental for objects with data (tables, sometimes some indexes)
- Are you sure DEVs always know best way to apply change to production table?
- Pattern-driven approach
- Declarative way vs procedural way
  - **Spoiler:** Declarative rockz!
- Rollback strategy

Two main questions to answer first

Your own requirements

When new column is added to the table, what do you expect to see in release package?

ALTER TABLE tbl ADD newColumn …

OR

DROP TABLE tbl
CREATE TABLE tbl (
…
newColumn…)

When new column is added to the table, how do you expect developer reflect this change in source code file of table in git?

ALTER TABLE tbl ADD newColumn …

OR

CREATE TABLE tbl (
…
newColumn…)

Matrix of possible implementation alternatives

In Release Package \ In Source Code	CREATE	ALTER
CREATE	Read git	*Weird!*
ALTER	Generate	Read git

Summary

- Efficient source code management is strong foundation for other capabilities
- Modular architecture is one of key enablers to incremental build, test and delivery
- On-demand environments possible even if you don’t use virtualization
- Testing automation is a challenge, but will pay off in long run
- Pattern-driven incremental package generation, rather than manual creation, will save you a lot of nerves and time
- … and keep in mind – DevOps is not only about automation ;)

Thank you.

Vladimir Filimonov

vladimir.filimonov@teradata.com