Evolutionary Database Design

SWE 4601 Software Design and Architecture

Based on 2016 article by Sadalage and Fowler

Pramod Sadalage

Martin Fowler

The Authors

Pramod Sadalage

Developed the original techniques of EDD and database refactoring used by ThoughtWorks in 2000. Co-authored the book about “Refactoring Databases”

Martin Fowler

Prominent author in the software engineering domain. Helps his colleagues explain what they've developed to the wider world of software development.

Remember this?
A software will either evolve or die

What EDD is about?

Is not a database design pattern

Is about how to maintain a database when the system is continuously evolving.

EDD is a design process, not the design itself.

A Scenario

- Story: user should be able to see, search, and update the location, batch, and serial numbers of a product in inventory.
- Assignee: Jen
- Current Status: A `inventory_code` field which is the concatenation of these three fields.
- To-do: split the field and store in three fields

What steps should done to implement this?

Checklist when migrating a database

- ✔ Schema is updated
- ✔ No data is lost
- ✔ Tests are updated
- ✔ Continuous integration is in place
- ✔ Deployment is automated

Jen's Steps

- Step 1: Write a migration script

```
ALTER TABLE inventory ADD location_code VARCHAR2(6) NULL;
ALTER TABLE inventory ADD batch_number VARCHAR2(6) NULL;
ALTER TABLE inventory ADD serial_number VARCHAR2(10) NULL;

UPDATE inventory SET location_code = SUBSTR(product_inventory_code,1,6);
UPDATE inventory SET batch_number = SUBSTR(product_inventory_code,7,6);
UPDATE inventory SET serial_number = SUBSTR(product_inventory_code,11,10);

DROP INDEX uidx_inventory_code;

CREATE UNIQUE INDEX uidx_inventory_identifier
  ON inventory (location_code,batch_number,serial_number);

ALTER TABLE product_inventory DROP COLUMN inventory_code;
```

- Step 2: Change any database code (views, stored procedures and triggers) to use the new columns
- Step 3: Change the application code to use the new columns
- Step 4: Run the script on a local database
- Step 5: Update tests

EDD Practices - Overview

1. DBAs collaborate closely with developers
2. All database artifacts are version controlled with application code
3. All database changes are migrations
4. Everybody gets their own database instance
5. Developers continuously integrate database changes
6. A database consists of schema and data
7. All database changes are database refactorings
8. Automate the refactorings
9. Developers can update their databases on demand
10. Clearly separate all database access code
11. Release frequently

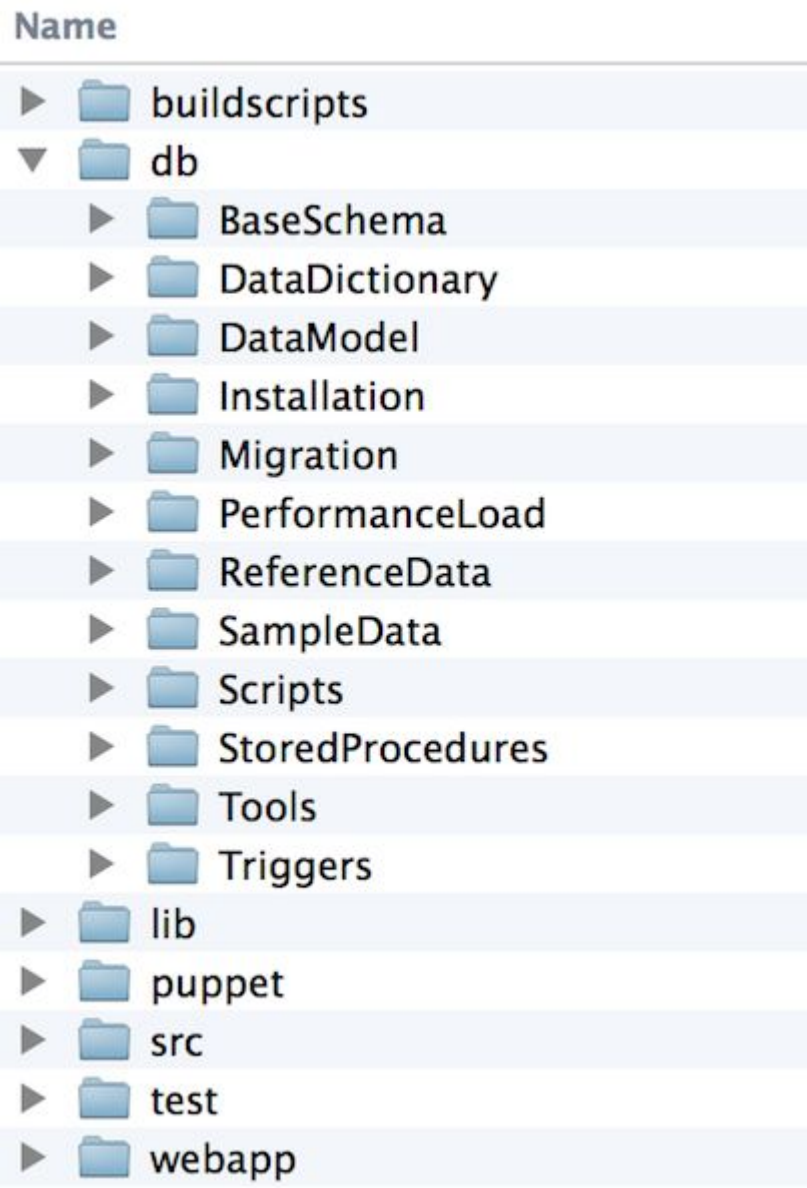
EDD Practice 1: DBAs collaborate closely with developers

- Is development task is going to make a significant change to the database schema?
- If so, the developer needs to consult with the DBA
  - The developer knows what new functionality is needed
  - DBA has a global view of the data in the application and other surrounding applications
- There must be organization-wide practice of collaboration

EDD Practice 2:
All database artifacts are version controlled with application code

Benefits:

- There's only one place to look
- Makes it easy to debug.
- Prevent deployments where the database is out of sync with the application.
- Makes it easy create new environments.



EDD Practice 3:
All database changes are migrations

- A common bad-practice:
  - DB change with tools or ad-hoc script
  - DBA compare the new DB version with old
  - DBA makes corresponding change in prod DB
- This approach is
  - Risky and error prone
  - Changes are not stored
  - Not in sync with application code
- EDD Practices
  - Changes are stored
  - In sync with app code

Migration Process

- ✔ Add unique IDs to each migration
- ✔ Track which migrations were applied
- ✔ Manage sequence constraint between migrations

```
select * from changelog;
```

24/24 Ln. 1 Col. 25 Lines: 1 INSERT WRITABLE \n UTF8 Delimiter: ;

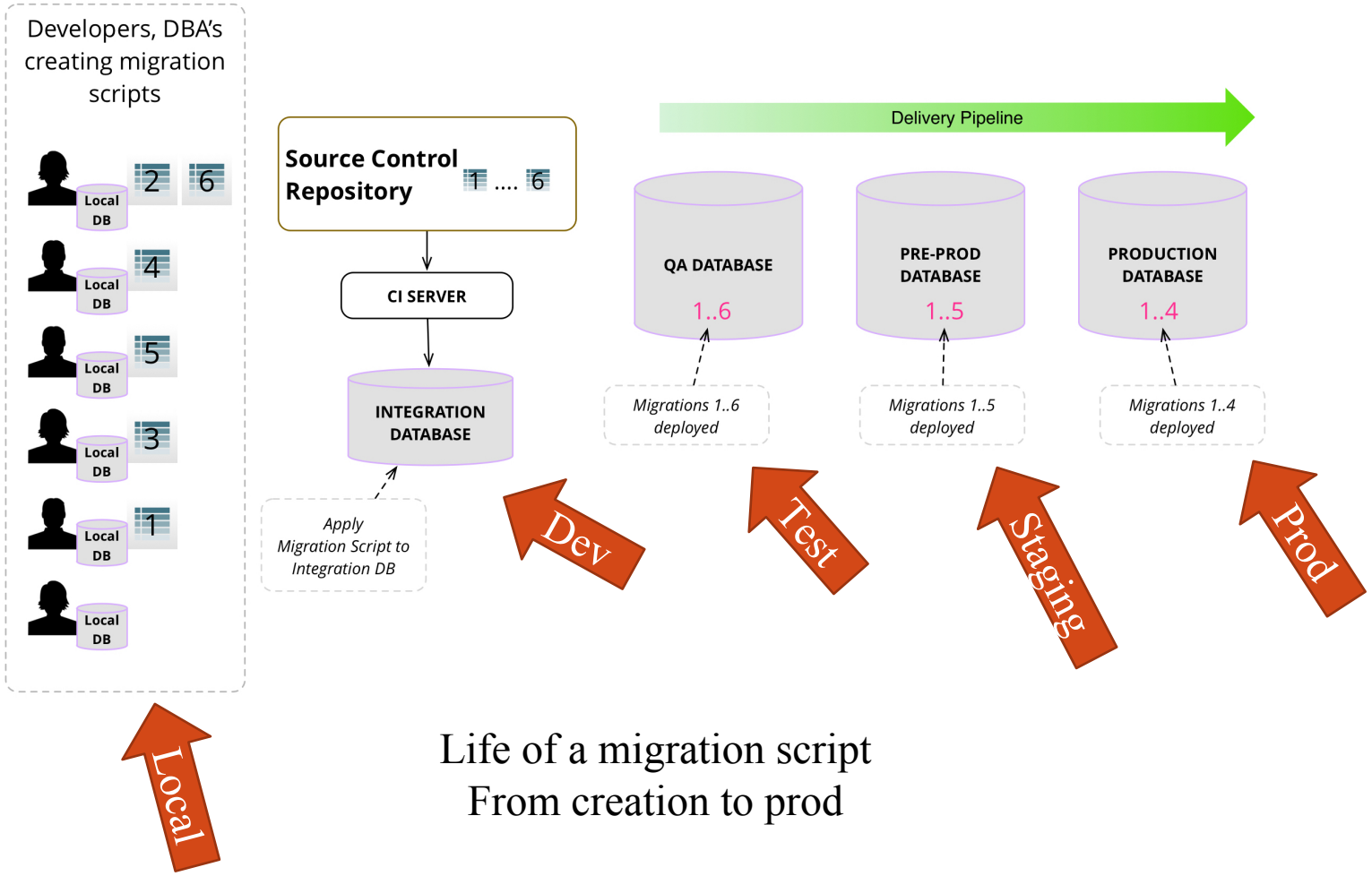
changelog

OFF

	CHANGE_NUMBER	COMPLETE_DT	APPLIED_BY	DESCRIPTION
1	1	2016-08-04 12:12:13.475	jen_dev	1_asset.sql
2	2	2016-08-04 12:12:13.525	jen_dev	2_asset_type.sql
3	3	2016-08-04 12:12:13.57	jen_dev	3_asset_parameters.sql
4	4	2016-08-04 12:12:13.586	jen_dev	4_inventory.sql
5	5	2016-08-04 12:12:13.628	jen_dev	5_split_inventory_code.sql
6	6	2016-08-04 12:12:13.671	jen_dev	6_equipment_type.sql
7	7	2016-08-04 12:12:13.717	jen_dev	7_add_insurance_value_to_equipment_type.sql
8	8	2016-08-04 12:12:13.723	jen_dev	8_data_location_equipment_type.sql

What if changes are done parallelly in branches?

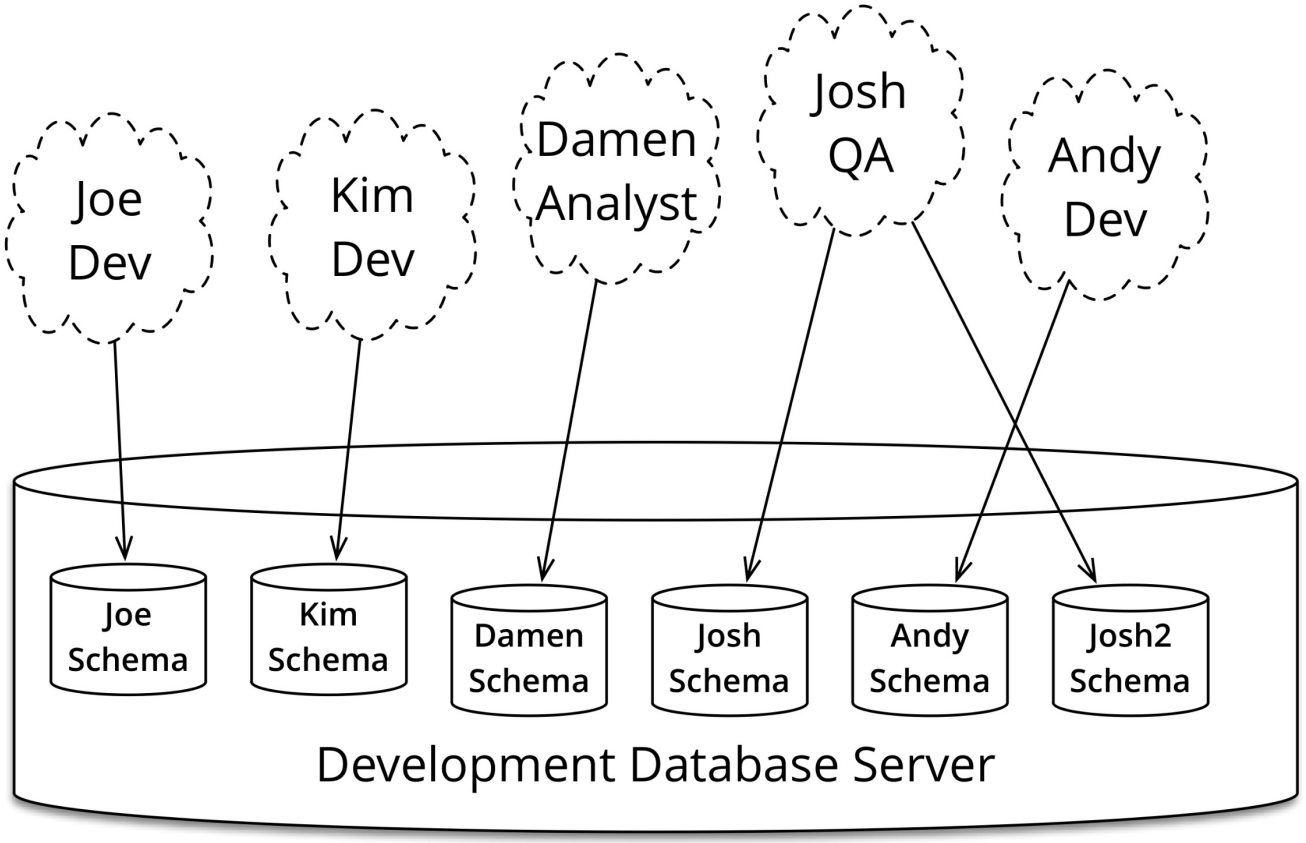
Migration Process (continued)



Life of a migration script
From creation to prod

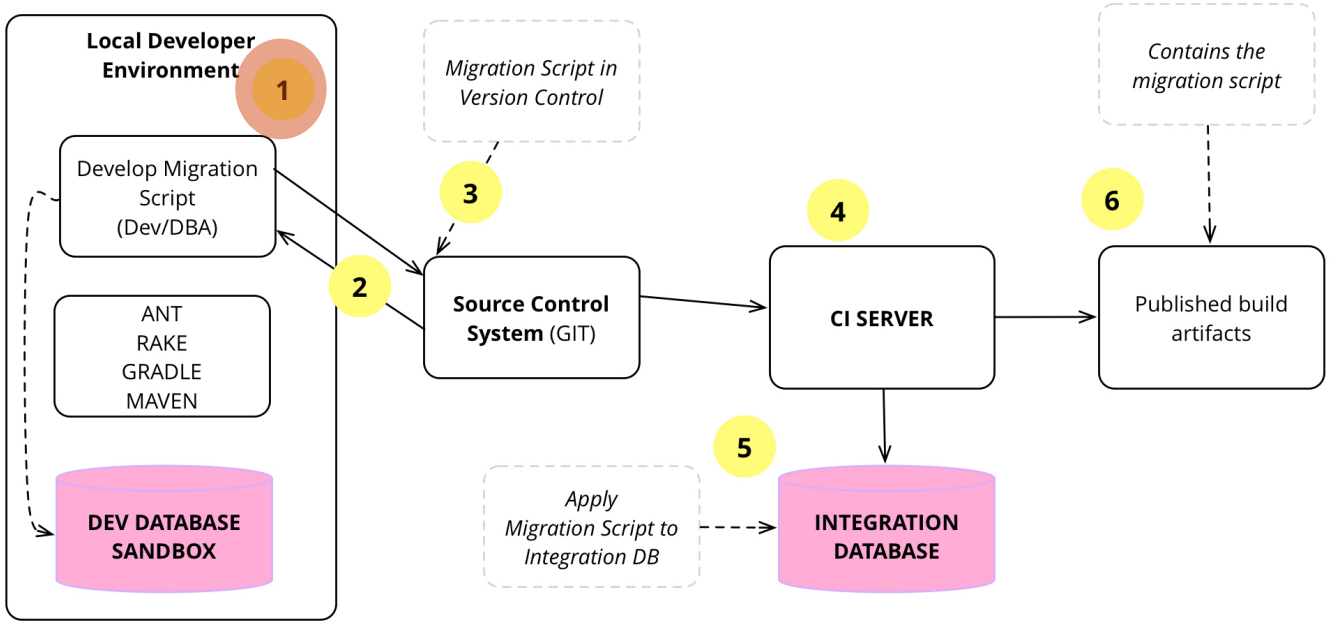
Can you map the environments?

EDD Practice 4: Everybody gets their own database instance



- Often considered difficult/impossible to manage
  - But proven useful in practice

EDD Practice 5:
Developers continuously integrate database changes



EDD Practice 6:
A database consists of schema and data

- Data includes
  - ✗ Application data
  - ✓ Static data (e.g. countries)
  - ✓ Sample data for testing

Which of the above data should be version controlled?

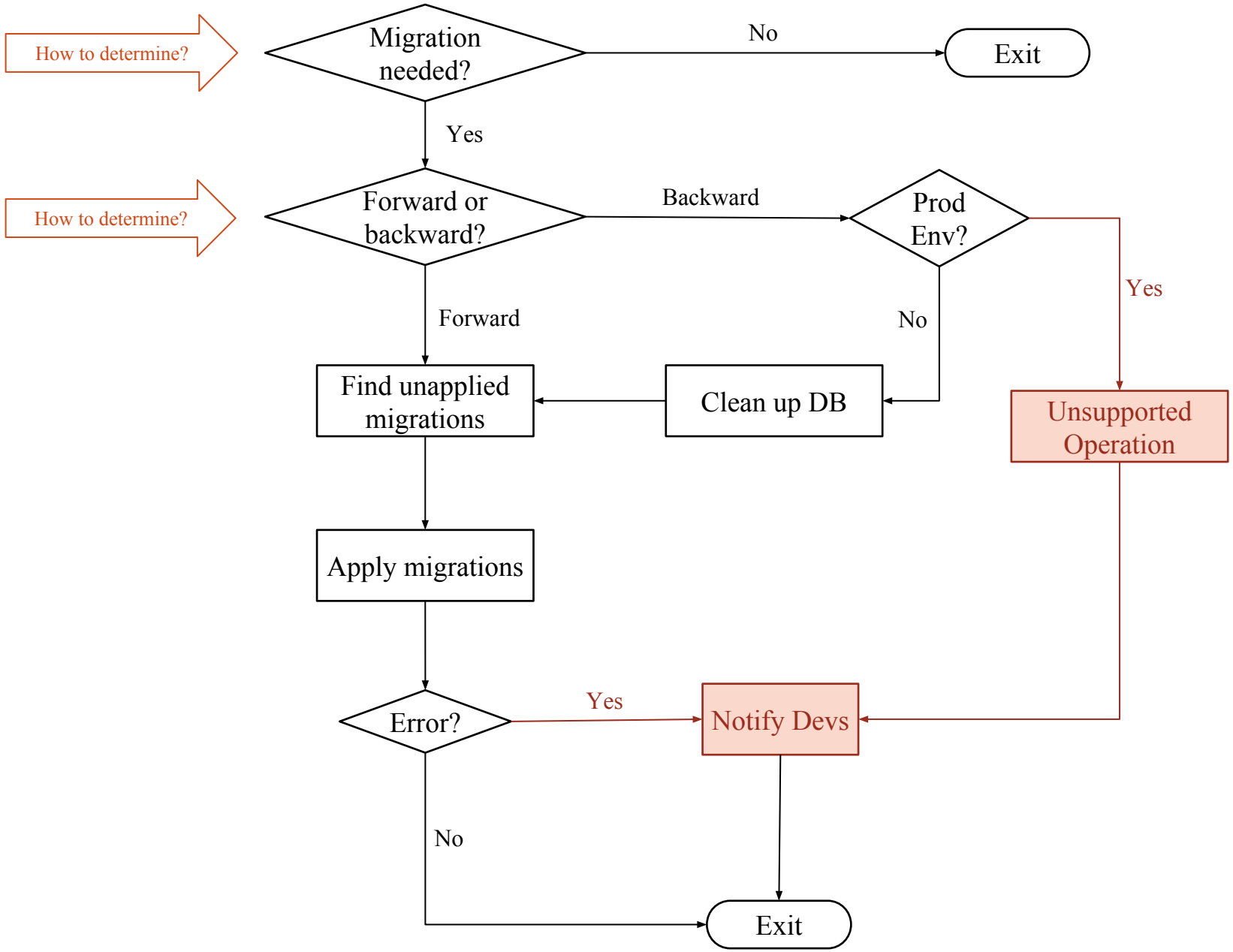
EDD Practice 9-11:

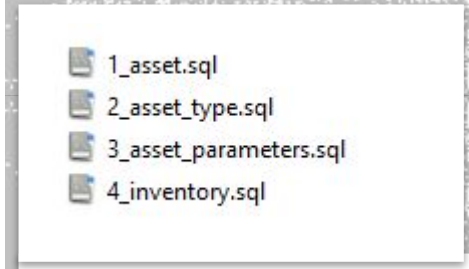
8 - 9 skipped.

9. Developers can update their database on-demand
10. Clearly separate all DB access code
11. Release frequently
    - Basically a common agile practice

Practices 7 and 8 are intentionally excluded

Automated Migration Process





DESCRIPTION
1_asset.sql
2_asset_type.sql
3_asset_parameters.sql
4_inventory.sql
5_split_inventory_code.sql
6_equipment_type.sql
7_add_insurance_value_to_equipment_type.sql
8_data_location_equipment_type.sql