# *CI/CD von relationalen Datenbanken*

**Jasmin Fluri**

schaltstelle
gemeinsam. selbstständig.

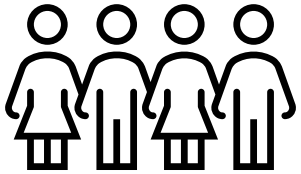
# *... eine kurze Geschichte einer Datenbank Änderung!*

Disclaimer
Personen und Prozesse sind frei erfunden.
Ähnlichkeiten mit echten Projekten sind zufällig und nicht beabsichtigt.

schaltstelle
gemeinsam. selbstständig.

# *In unserem imaginären Entwicklungsprojekt ....*

... bestehend aus einer Spring Boot Applikation...



...arbeiten 4 Entwickler*innen!

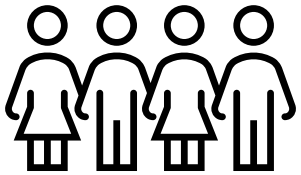
.... und einer Oracle Datenbank ...

schaltstelle

# *In unserem Entwicklungsprojekt ....*

CI/CD der Spring Boot App ist durchgängig aufgesetzt!
Testing ist fester Bestandteil!

Bei der Datenbank sieht das aber anders aus...



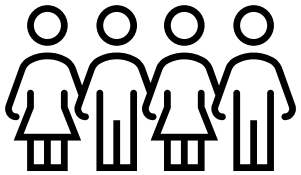
Niemand im Team ist von Anfang an dabei, der Code wurde jeweils von Vorgänger*innen übernommen.

Die Skills liegen vor allem in der Entwicklung der Spring Boot App.

schaltstelle

# *In unserem Entwicklungsprojekt ….*

… wird diese versucht entweder applikationsseitig zu abstrahieren…



…oder komplett zu vermeiden!

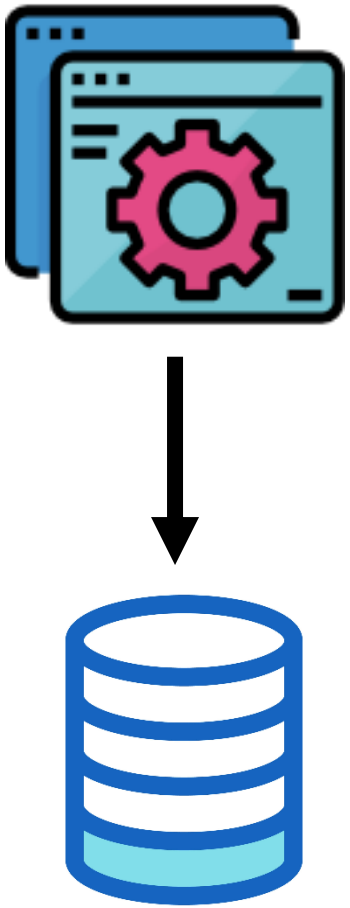
Falls es doch unvermeidbar ist…

Bei einer aufkommenden Datenbank-Änderung …

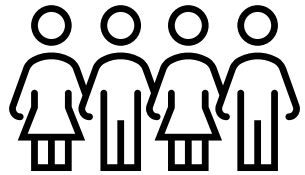
schaltstelle

# *In unserem Entwicklungsprojekt ....*

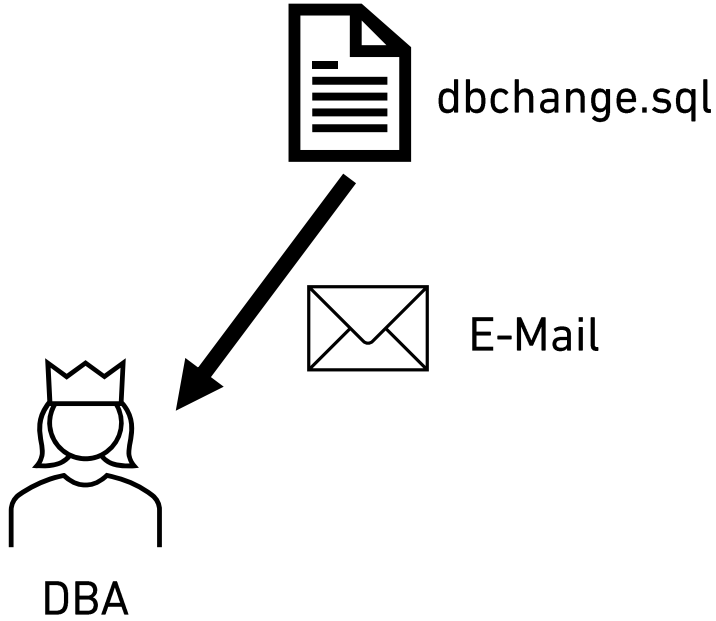
Die Ausführung ändert die Zugriffslogik zur Applikation, weshalb diese neu deployed werden muss.



Die DBA führt das Script manuell aus!



... muss jemand im Team ein Migrationsscript schreiben!




schaltstelle

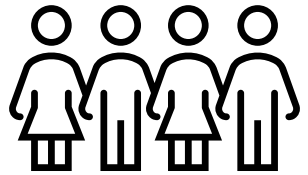
# *.. Weshalb ist dieses Vorgehen problematisch?*

schaltstelle
gemeinsam. selbstständig.

# *In unserem Entwicklungsprojekt ....*

Die Ausführung ändert die Zugriffslogik zur Applikation, weshalb diese neu deployed werden muss.

Kein versionierter Zugriffslayer, was zu Downtime führt!



... muss jemand im Team ein Migrationsscript schreiben!

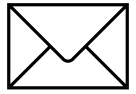
SQL Scripts sind nicht unter VCS!

dbchange.sql

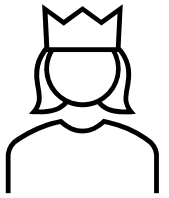
Keine Regressionstests auf Seiten DB!

Die DBA führt das Script manuell aus!

Manuelle Ausführung lässt viel Raum für Fehler!



E-Mail



DBA

Weil per Email versendet, wissen wir nicht genau, wann der Change gemacht wird!

schaltstelle

# *Über mich*

## Jasmin Fluri

Database / Automation Engineer
Schaltstelle GmbH

«The person that builds a CI/CD pipeline in every new project.»

- *Database Development*
- *Development Process*
- *Automation and Tooling*
- *Continuous Integration*
- *Continuous Delivery*
- *Software Engineering*
- *Pipelining / Provisioning*

@jasminfluri

@jasminfluri

jasminfluri

schaltstelle

Oracle ACE Pro

SYMPOSIUM

# *Agenda*

- Story DB Change
- Über mich
- Effekte ohne CI/CD
- Software Releases
- Datenbankentwicklung in Zahlen
- CI/CD Applikation vs. DB Applikation
- Vorbedingungen für DB CI/CD
- Versionskontrolle
- Statische Code Analyse
- DB Migrations Tool
- Automatisierte Tests
- Decoupled Codebase
- Effekte mit CI/CD
- Q/A
- Social Part

schaltstelle

# *Was geschieht wenn DB Changes vermieden werden?*

schaltstelle
gemeinsam. selbstständig.

Wenn wir unsere (DB)-Applikationen vernachlässigen, zerfallen diese!

… it appears that most large enterprises actually care about **minimizing application maintenance** of existing production systems. That causes them to utilize “bad” schemas, and generally to **allow “database decay”.**

– *Stonebreaker et al. (2016)*

schaltstelle

2016 IEEE International Conference on Big Data (Big Data)

# Database Decay and How to Avoid It

Michael Stonebraker Dong Deng Michael L. Brodie
*M.I.T. Computer Science and Artificial Intelligence Laboratory*
{stonebraker,dongdeng,mlbrodie}@csail.mit.edu

***Abstract*—The traditional wisdom for designing database schemas is to use a design tool (typically based on a UML or E-R model) to construct an initial data model for one's data. When one is satisfied with the result, the tool will automatically construct a collection of 3rd normal form relations for the model. Then applications are coded against this relational schema. When business circumstances change (as they do frequently) one should run the tool again to produce a new data model and a new resulting collection of tables. The new schema is populated from the old schema, and the applications are altered to work on the new schema, using relational views whenever possible to ease the migration. In this way, the database remains in 3rd normal form, which represents a "good" schema, as defined by DBMS researchers. "In the wild", schemas often change once a quarter or more often, and the traditional wisdom is to repeat the above exercise for each alteration.**

**In this paper we report that the traditional wisdom appears to be rarely-to-never followed for large, multi-department applications. Instead DBAs appear to attempt to minimize application maintenance (and hence schema changes) instead of maximizing schema quality. This leads to schemas which quickly diverge from E-R or UML models and actual database semantics tend to drift farther and farther from 3rd normal form. We term this divergence of reality from 3rd normal form principles *database decay*. Obviously, this is a very undesirable state of affairs, and should be avoided if possible.**

**The paper continues with tactics to slow down database decay. We argue that the traditional development methodology, that of coding applications in ODBC or JDBC, is at least partly to blame for decay. Hence, we propose an alternate methodology that should be more resilient to decay.**

## 1. Introduction

There has been significant on-going research into schema design methodologies for at least the 40 years since Peter Chen wrote his pioneering paper [1]. There has been work on design paradigms [2], [3], schema evolution [4], [5], model management [6], [7], and reports on a variety of commercial offerings [8], [9], [10]. In our opinion, much of this work is misguided, as we explain in this paper, because it focuses on the wrong metrics. Specifically, the research work focuses on constructing and maintaining "good" schemas. Instead, it appears that most large enterprises actually care about minimizing application maintenance of existing production systems. That causes them to utilize "bad" schemas, and generally to allow "database decay". Our assertion is based on conversations with nearly twenty Database Administrators (DBAs) at three very large enterprises.

The purpose of this paper is to explain why decay occurs, and then to explore tactics that minimize database decay. In our opinion, the most significant contributing factor is the development methodology that is traditionally employed in large organizations, that of coding applications using ODBC/JDBC. Hence, we propose a different methodology that is more resilient to decay.

The rest of the paper is organized as follows. In Section 2 we explore the environment we see in most large enterprises. This serves to frame the reasons for database decay, which we explain in Section 3. Sections 4 and 5 then turn to antidotes for decay. In Section 4 we consider **defensive** database design and application construction, and show how this can help. Section 5 then explores a design paradigm that is more resilient to decay.

## 2. The Design Environment in Large Organizations

### 2.1. Longevity

In this section we explain the environment found in most large organizations. First, as has been pointed out by many authors, databases last for a long time. What is less well known is the frequency with which business conditions force them to change. New applications for the same data, changed business requirements for the existing applications and mergers and acquisitions (M&A) all contribute to frequent changes. As a figure of merit, we assume databases change once per quarter, although the data in [11] suggests it is even more frequent.

### 2.2. Decentralization

Researchers often assume that a database is controlled by a DBA, who is embedded in the application group that is writing or maintaining the applications that access a given database, as noted in Figure 1. We call this model "centralization" as all aspects of database design and deployment are controlled in one place. Although there are examples of this organization, especially with small-to-midsize applications,

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

*oft hört man auch...*

*«... wir vermeiden DB Changes nicht, aber wir haben fixe Release Windows!»*

schaltstelle
gemeinsam. selbstständig.

# *Welche Effekte haben fixe Release Windows?*

schaltstelle
gemeinsam. selbstständig.

# *Mit fixen Release Windows ...*

**Sind Features fertig und müssen warten...**

- ... dies verursacht Abhängigkeiten unter Features.

**Risiko nimmt zu, viele Features auf einmal zu deployen**

- ... Releases werden grösser und grösser!

**Kurze Feedback Zyklen sind nicht möglich!**

- ... Feedback gibt es erst beim Release Window.

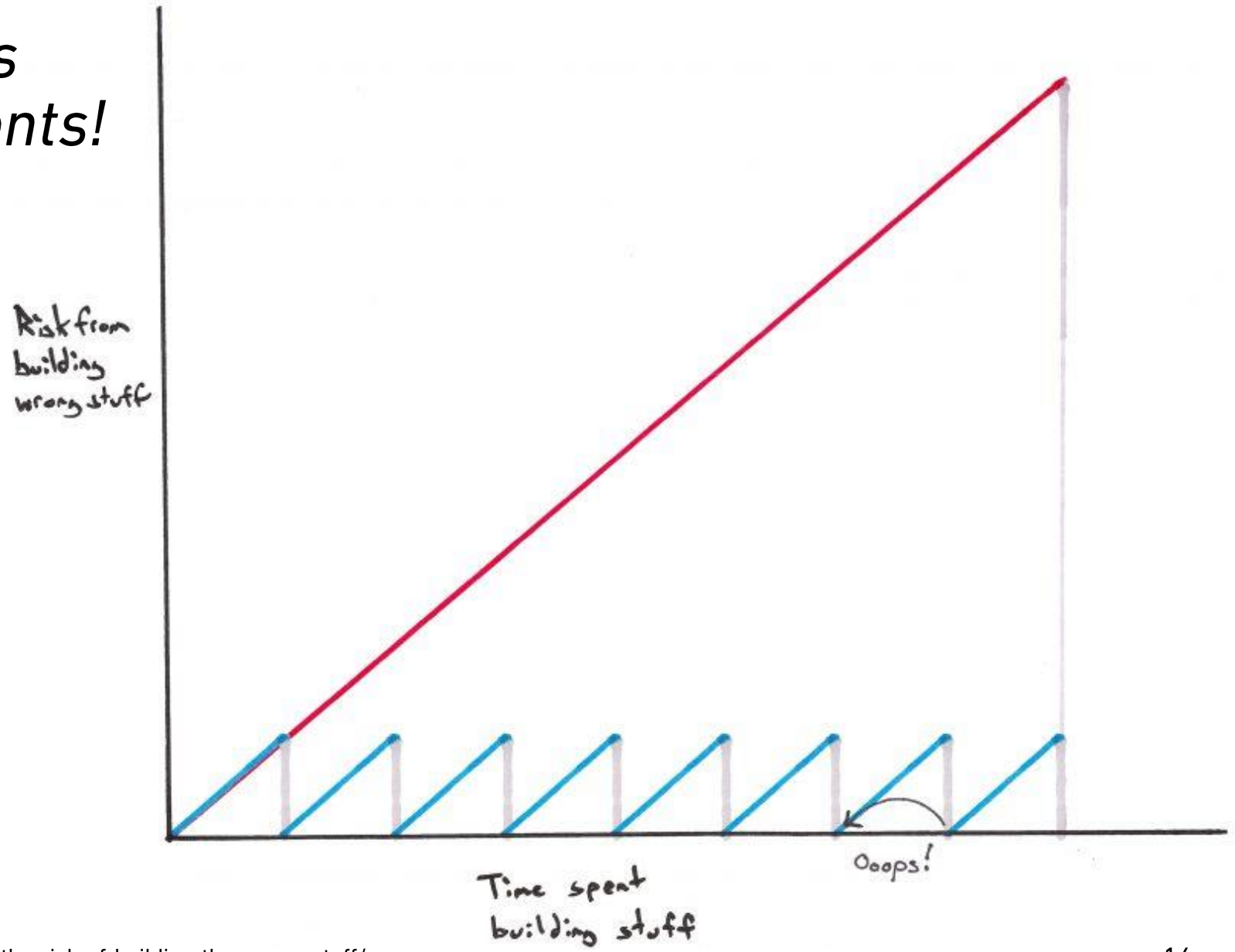
schaltstelle

*Je länger wir warten, umso grösser wird das Risiko eines Deployments!*

Features werden gekoppelt!

Mental Load bei Developern steigt!

Risiko steigt!




schaltstelle

Source: https://kromatic.com/blog/the-risk-of-building-the-wrong-stuff/

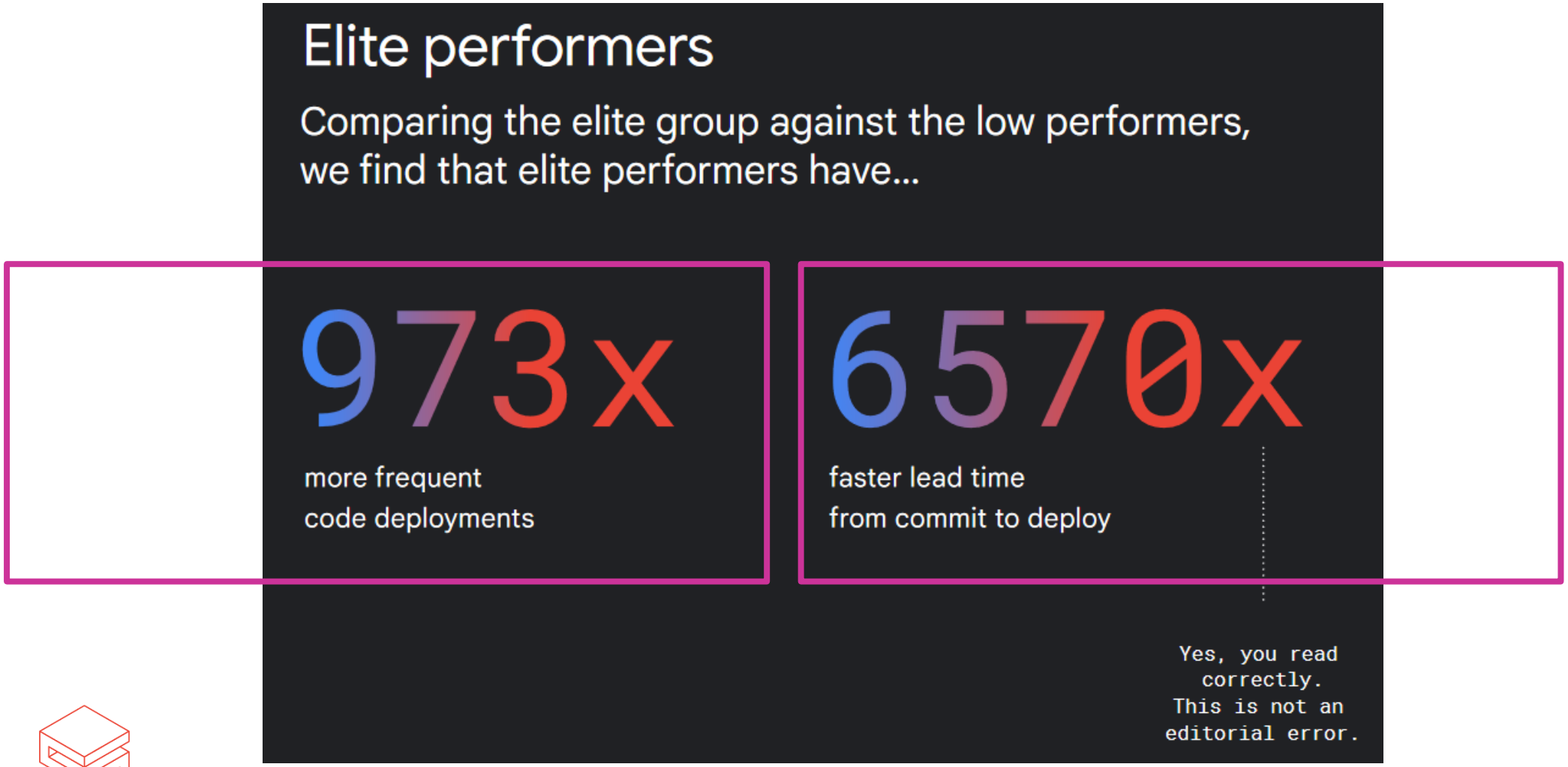
*Wir sollten Features ausliefern, wenn sie fertig sind! Nur in der Produktion wissen wir, wie sie sich in der Realität verhalten.*

schaltstelle
gemeinsam. selbstständig.

# *Was wissen wir über das Releasen von Software?*

schaltstelle
gemeinsam. selbstständig.

# *Effiziente Teams deployen Changes mindestens einmal pro Tag!*




Source: DevOps Report DORA 2021

schaltstelle

# *Effiziente Teams haben eine tiefe Change-Fehlerquote!*

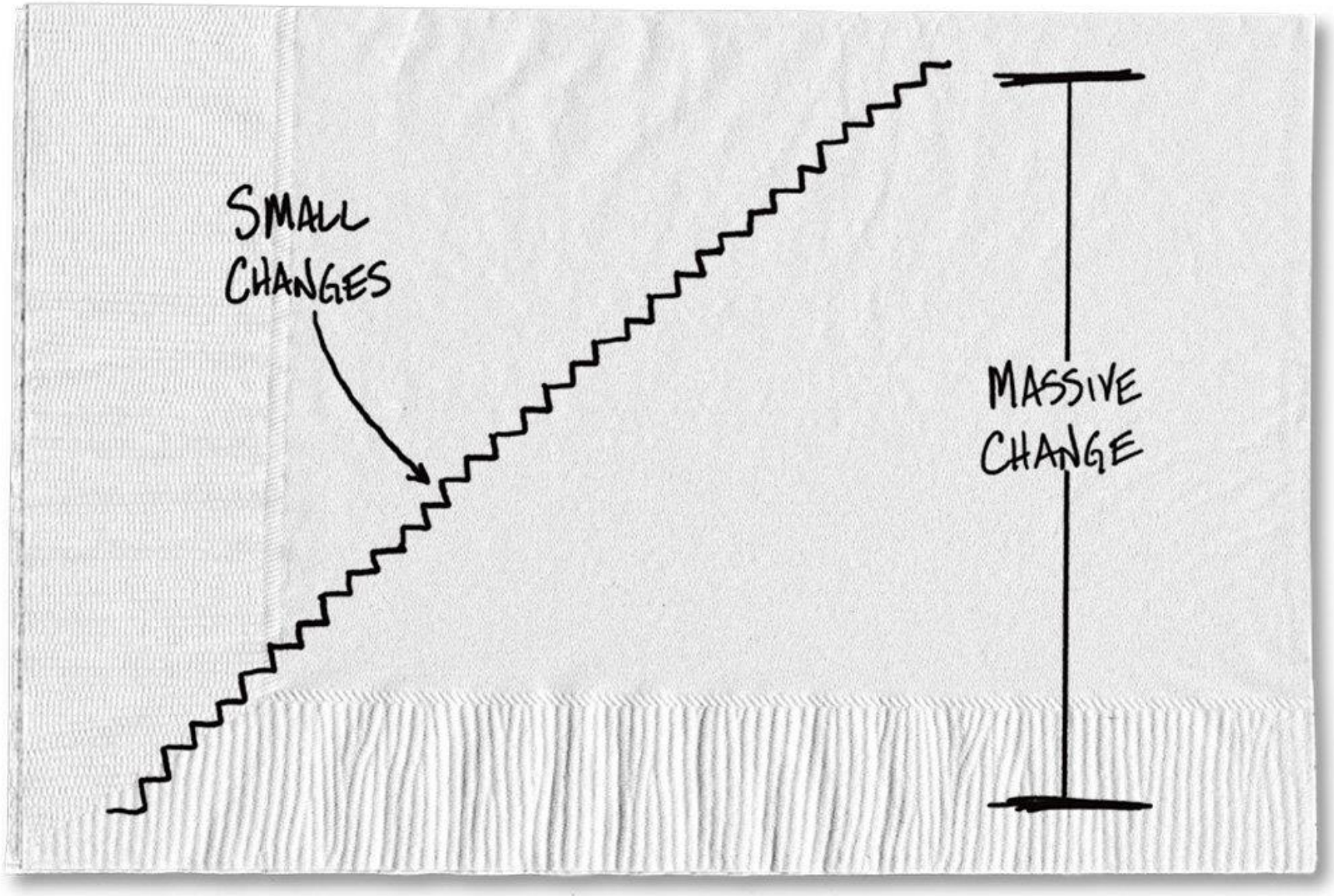

Source: DevOps Report DORA 2021

schaltstelle

*Warum erholen sie sich so viel besser von Zwischenfällen?*

schaltstelle
gemeinsam. selbstständig.

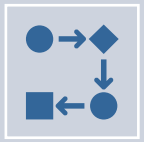
# *Kleine Changes = kleines Risiko = kleiner Impact = kleine Feedback Loops*




schaltstell

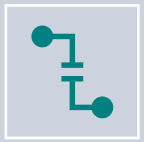
Source: NYTimes – The best path to long term success is slow, simple and boring!

# *Welche positiven Effekte haben kleine Changes?*



Kleine Änderungen haben kleine Auswirkungen! Es ist einfach, alle Elemente zu sehen, die betroffen sind!

Kleine Änderungen bergen nur ein geringes Risiko!



Kleine Änderungen können leichter rückgängig gemacht oder korrigiert werden, wenn sie fehlerhaft sind.

schaltstelle

# *Ziel von Software Releases:*

*We want to ship small changes continuously to get fast and continuous feedback!*

schaltstelle
gemeinsam. selbstständig.

*… kleine Changes zu deployen sind somit eine gute Praktik!*

schaltstelle
gemeinsam. selbstständig.

*… aber in der Datenbankentwicklung ist es nicht sehr verbreitet!*

schaltstelle
gemeinsam. selbstständig.

# *Datenbankentwicklung in Zahlen!*

schaltstelle
gemeinsam. selbstständig.

*Mehr als die Hälfte aller Datenbankapplikationen besitzen keine Datenqualitäts- oder Applikationstests!*

schaltstelle
gemeinsam. selbstständig.

*Weniger als die Hälfte der DB-Entwicklungsprojekte besitzt automatisierte Entwicklungs-Workflows!*

schaltstelle
gemeinsam. selbstständig.

*Fehlende Automatisierung von DB-CI/CD ist eines der häufigsten Bottlenecks im Releasing.*

schaltstelle
gemeinsam. selbstständig.

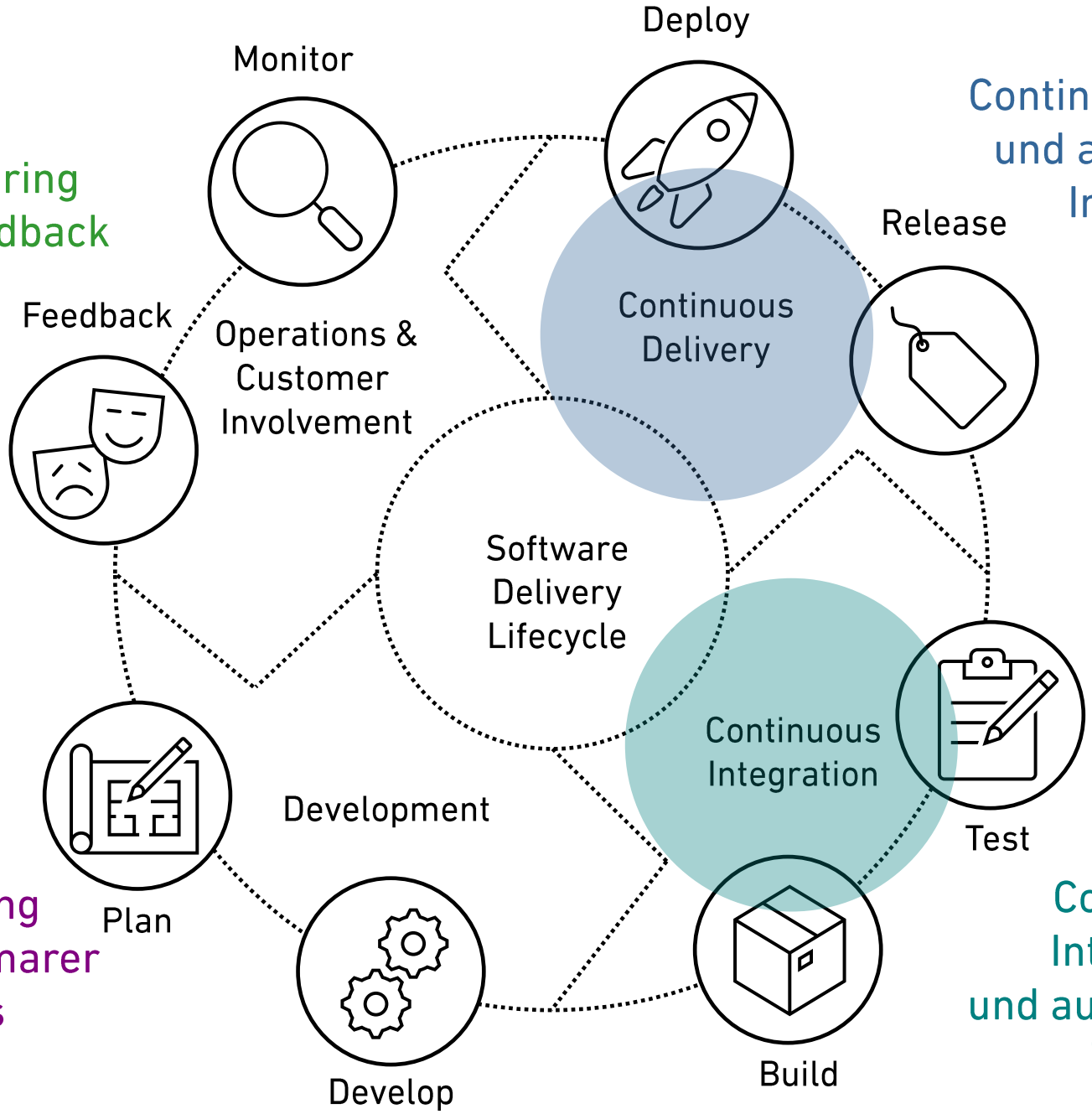
*Weniger als 30% der DB-Entwicklungsprojekte besitzen sowohl automatisierte Tests wie auch Statische Code Analyse.*

schaltstelle
gemeinsam. selbstständig.

# *Was verstehen wir unter CI/CD?*

schaltstelle
gemeinsam. selbstständig.

# *Software Delivery Lifecycle*




Monitoring
und Feedback

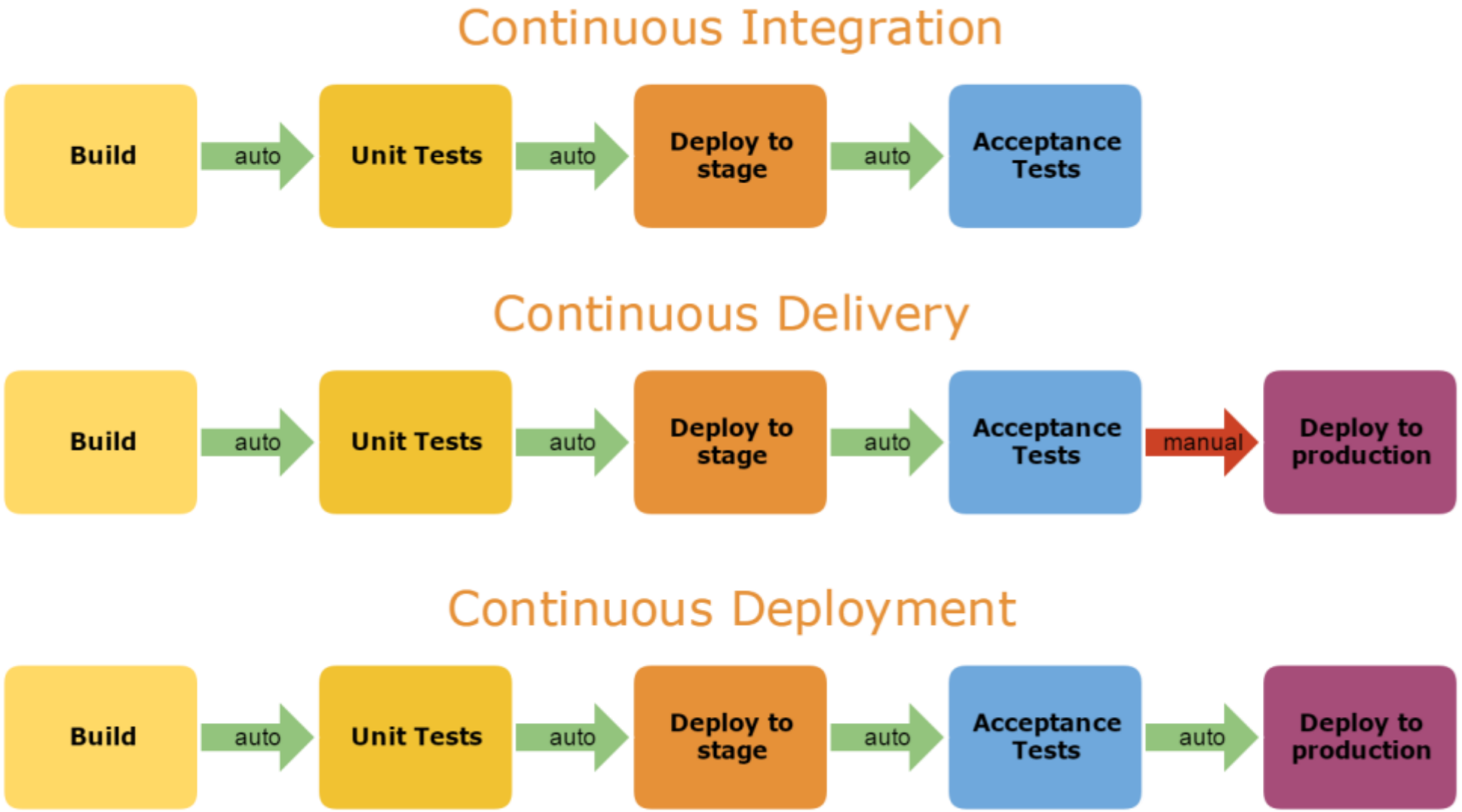
Continuous Releasing
und automatisierte
Installation

Entwicklung
kleiner / atomarer
Changes

Continuous
Integration
und automatisiertes
Testing

schaltstelle

# *Continuous Delivery != Continuous Deployment*

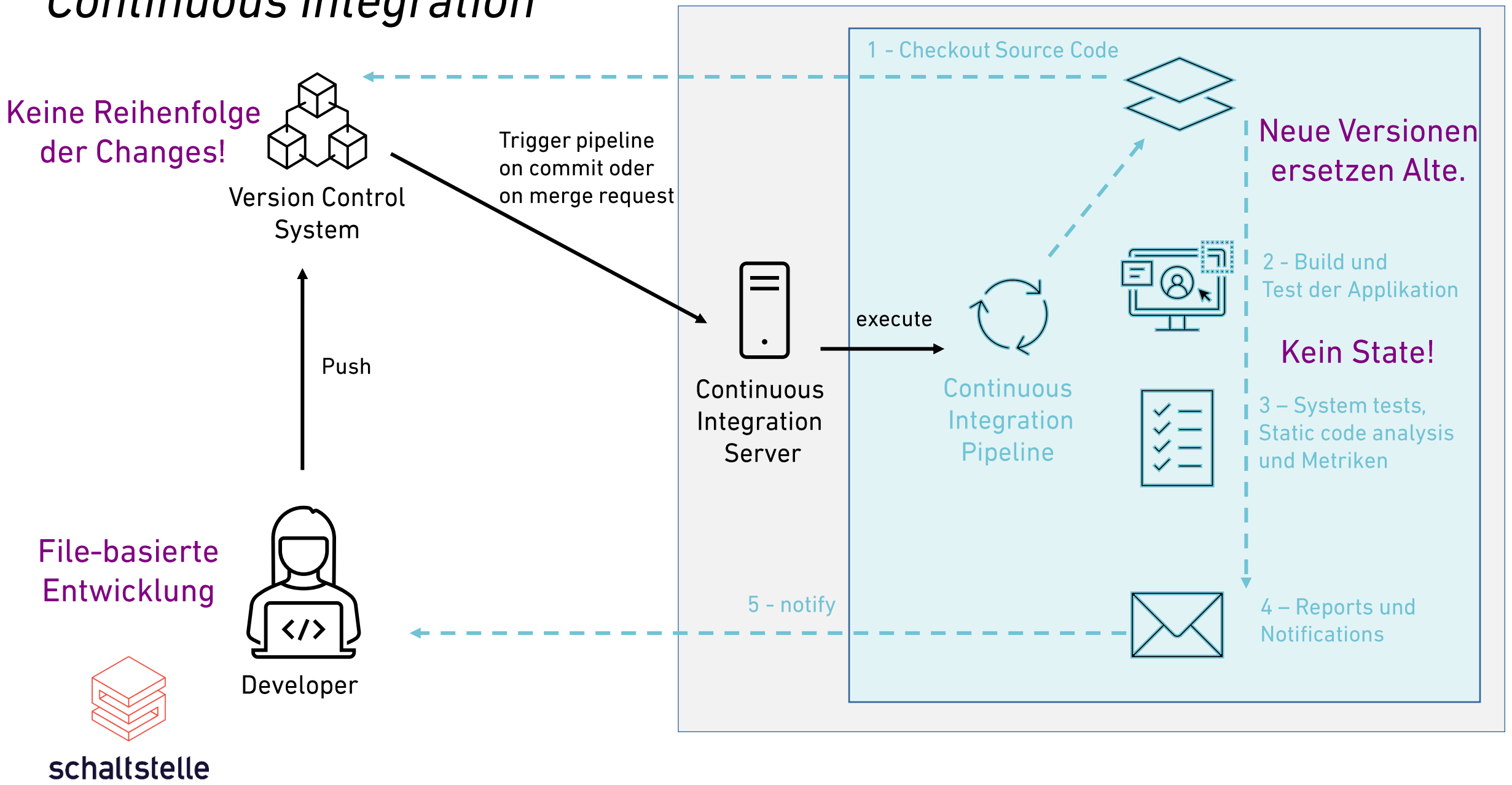



schaltstelle

# *Wie ist CI/CD für Applikationen aufgebaut (non-DB)?*

schaltstelle
gemeinsam. selbstständig.

# *Continuous Integration*



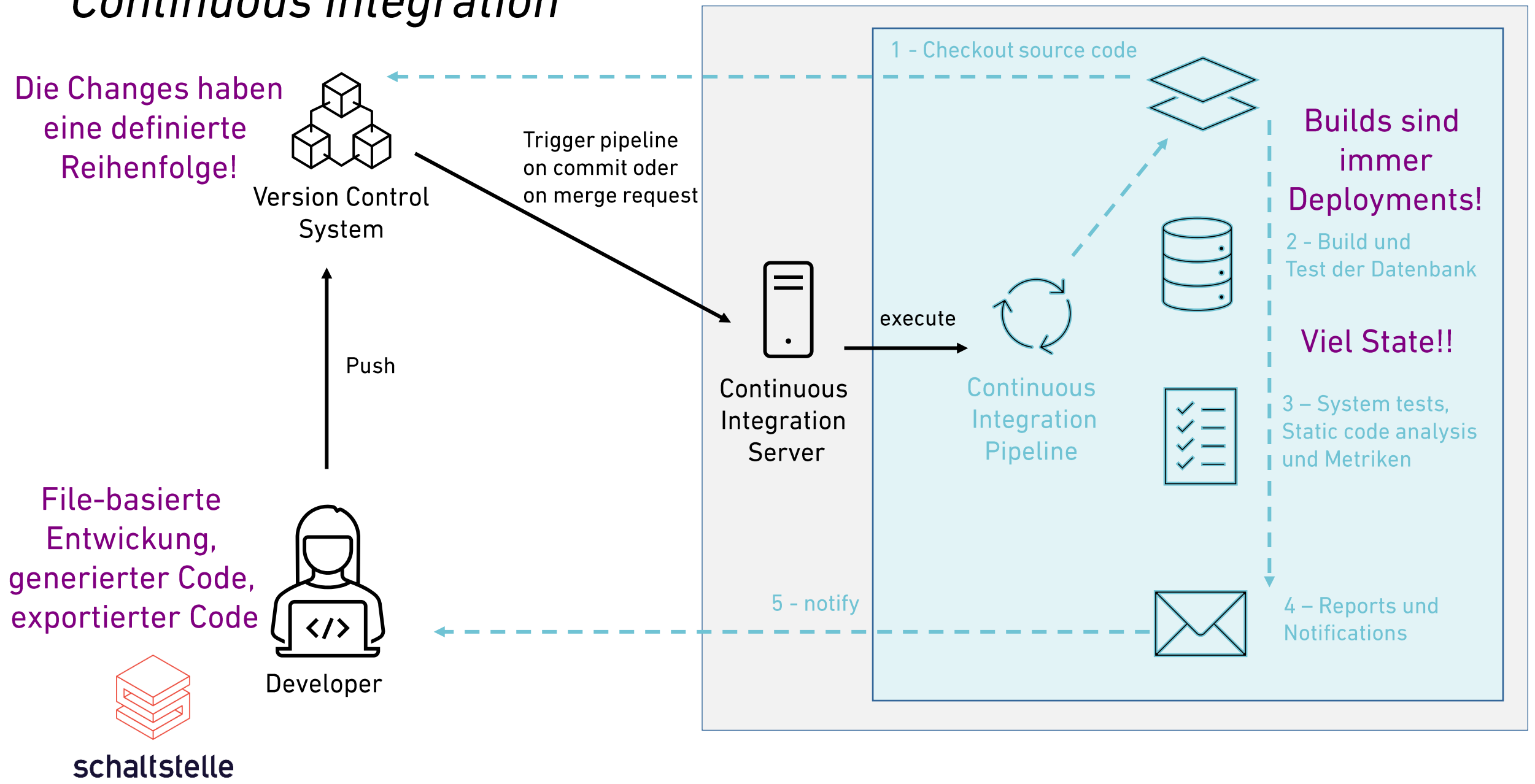
# *Wie unterscheidet sich Datenbank CI/CD?*

schaltstelle
gemeinsam. selbstständig.

```
if (system = database){
    build == deployment;
}
```

schaltstelle
gemeinsam. selbstständig.

# *Continuous Integration*

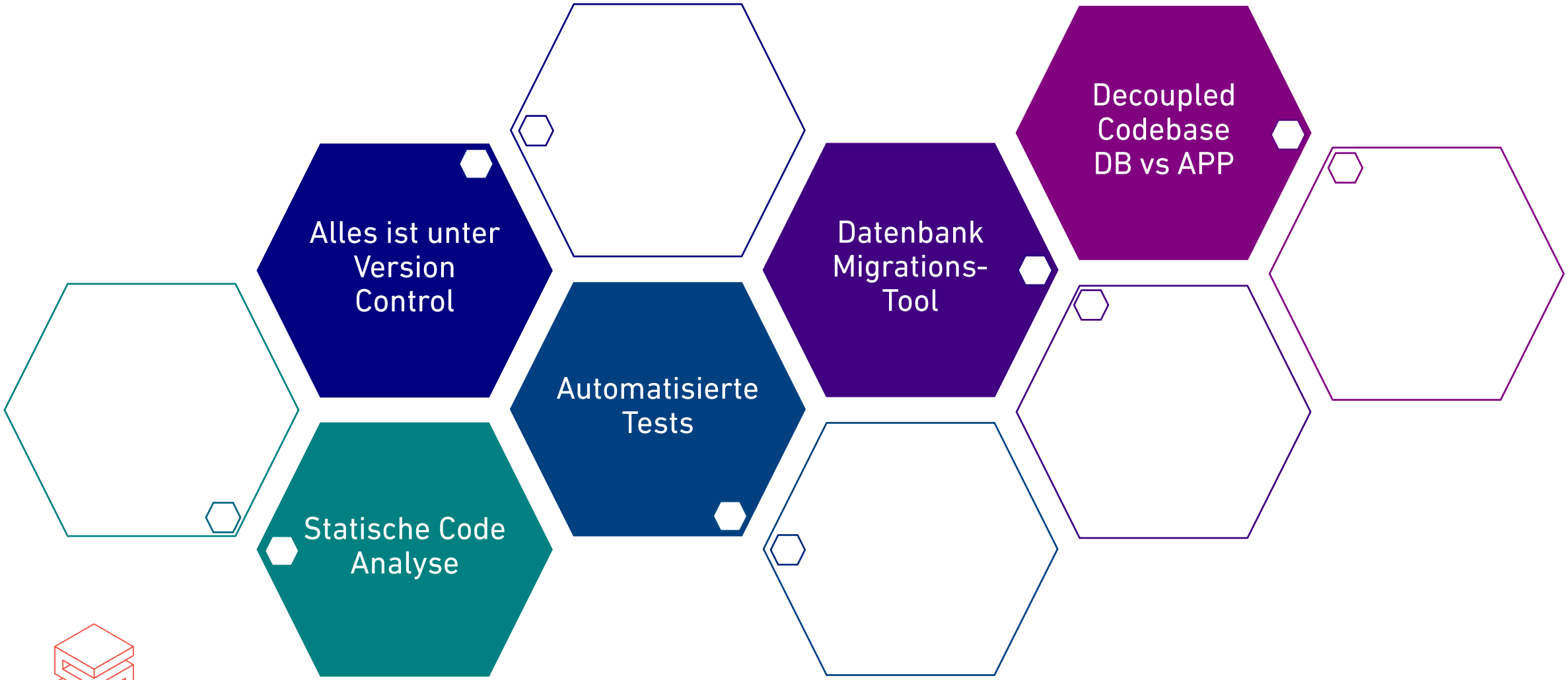



schaltstelle

# *Was sind Vorbedingungen für ein gutes Datenbank CI/CD?*

schaltstelle
gemeinsam. selbstständig.

# *Before you start building a database CI/CD pipeline you need...*




schaltstelle

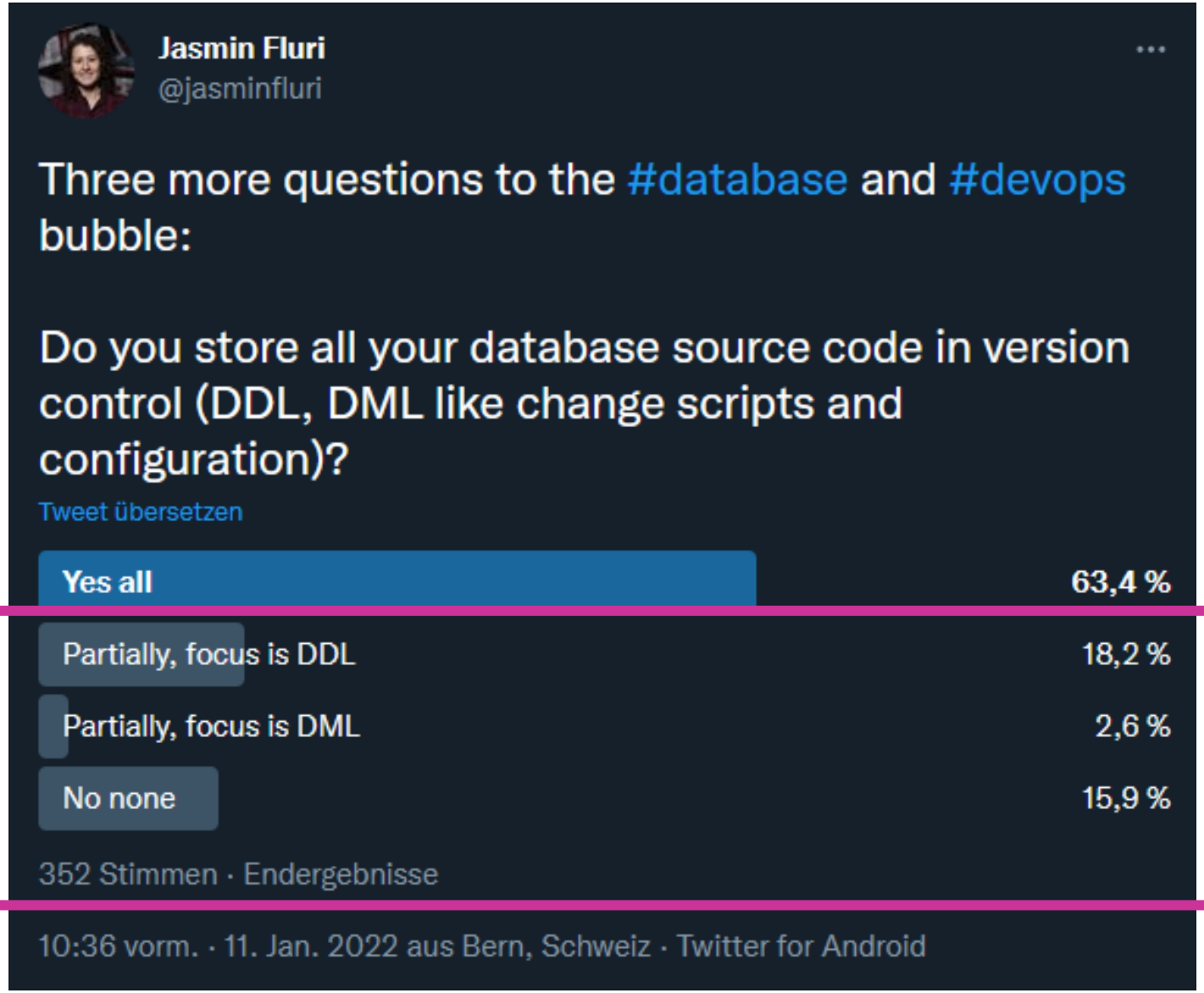
# *(1) Versionskontrolle*

schaltstelle
gemeinsam. selbstständig.

*... without Version Control there's no single source of truth!*

schaltstelle
gemeinsam. selbstständig.

# *Alles was zur DB-Applikation gehört, muss unter Versionskontrolle!*




36.6%

schaltstelle

# *Datenbank Source Code in der Versionskontrolle*

## Beispiel Projektsturktur

```
myproject/
├── …
├── docs/
│   └── documentation.md
├── db/
│   ├── 0_sequences/
│   ├── 1_tables/
│   ├── 2_mviews/
│   ├── 3_views/
│   ├── 4_packages/
│   └── 5_utils/
├── migrations/
│   └── scripts/
└── db-tests/
    ├── packages/
    └── data/
```

DDL Code

Migration Scripts

Test Code und Test Daten

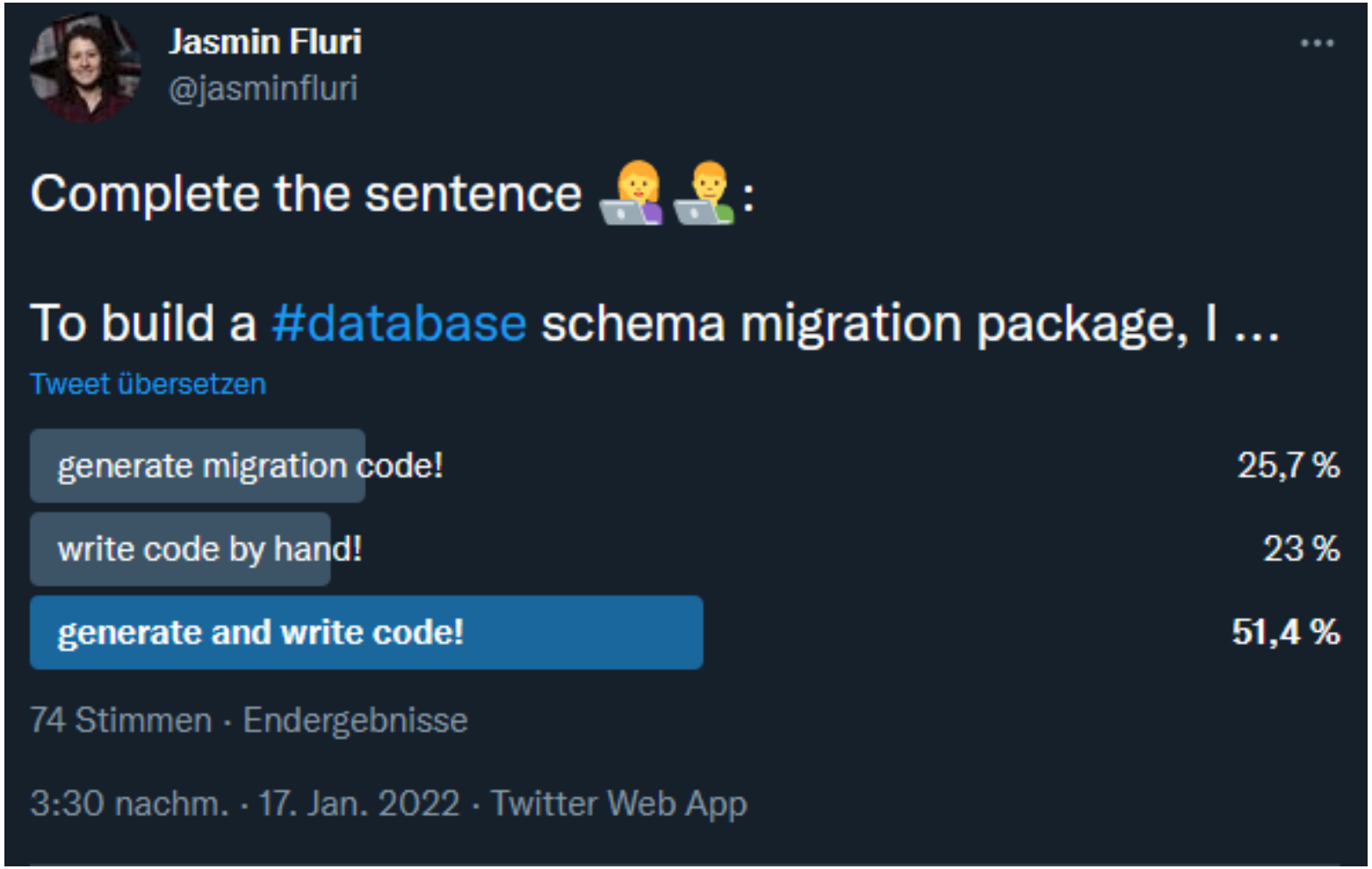
- DDLs enthalten die Objektdefinitionen!
- Migration Scripts ermöglichen das Upgrade auf die nächste Version
- Test Code testet unsere Datenbank Logik und ihr Verhalten!

schaltstelle

*ABER: Reines File-Based Development ist in der DB-Entwicklung unüblich.*

schaltstelle
gemeinsam. selbstständig.

*The majority of people uses a mix of generating and writing migration scripts!*




schaltstelle

# *Versionskontrolle: wir brauchen DDLs und Migrations-Scripte!*

Das Speichern von Migration Scripts alleine reicht meistens nicht, man benötigt auch die DDLs im VCS!

Mit den DDLs lässt sich ein System schneller neu bauen!

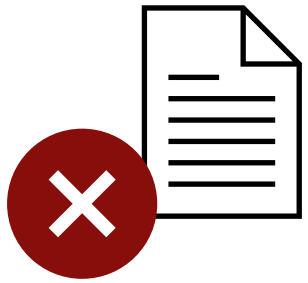
Migrations-Scripte ermöglichen es von der einen Version zur nächsten zu migrieren!

schaltstelle

*Damit wir robuste Deployments erreichen können, müssen unsere Migrations-Scripts repeatable sein!*

schaltstelle
gemeinsam. selbstständig.

# *Was passiert, wenn Migrations-Scripts nicht repeatable sind?*



**Nicht wiederholbares Migrationsskript**
Wenn es fehlschlägt, brauchen wir ein neues Skript mit den verbleibenden Änderungen (und Korrekturen). Ansonsten würde das Script sofort auf Fehler laufen.

**Wiederholbares Migrationsskript**
Wenn es fehlschlägt, ändern wir es und führen es erneut aus - es wird dort fortgesetzt, wo es fehlgeschlagen ist.

schaltstelle

# *Passive Versionskontrolle*

*(Code aus der DB generieren oder exportieren) birgt das Risiko, dass wir es versäumen, Dinge zu exportieren, die wir geändert haben.*

schaltstelle
gemeinsam. selbstständig.

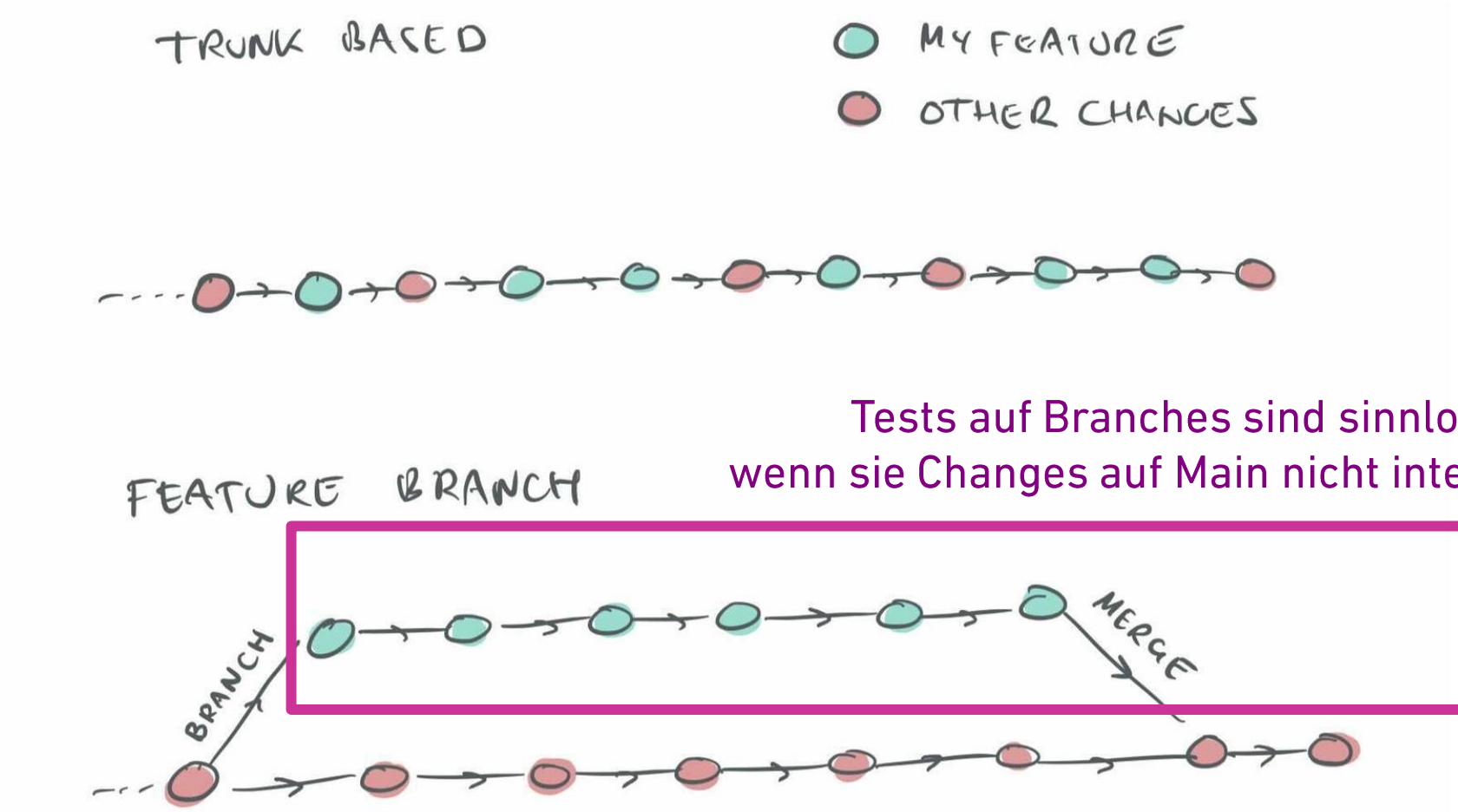
# *Branching-Strategien in der DB Entwicklung!*

schaltstelle
gemeinsam. selbstständig.

*DB-Changes sind verwandt mit Infrastruktur-Changes. Oftmals bauen sie aufeinander auf und machen nur in Sequenz Sinn.*

schaltstelle
gemeinsam. selbstständig.

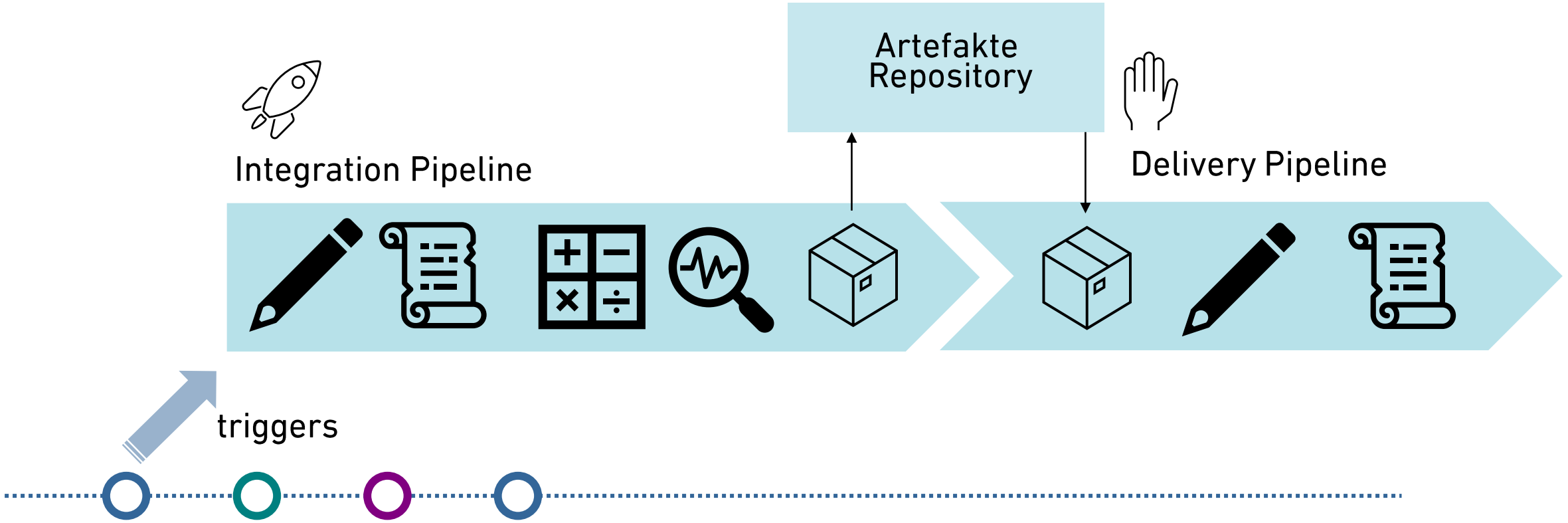
# *Das Problem mit Feature Branches*




Tests auf Branches sind sinnlos,
wenn sie Changes auf Main nicht integrieren!

schaltstelle

# *Trunk-based development and Continuous Integration*




schaltstelle

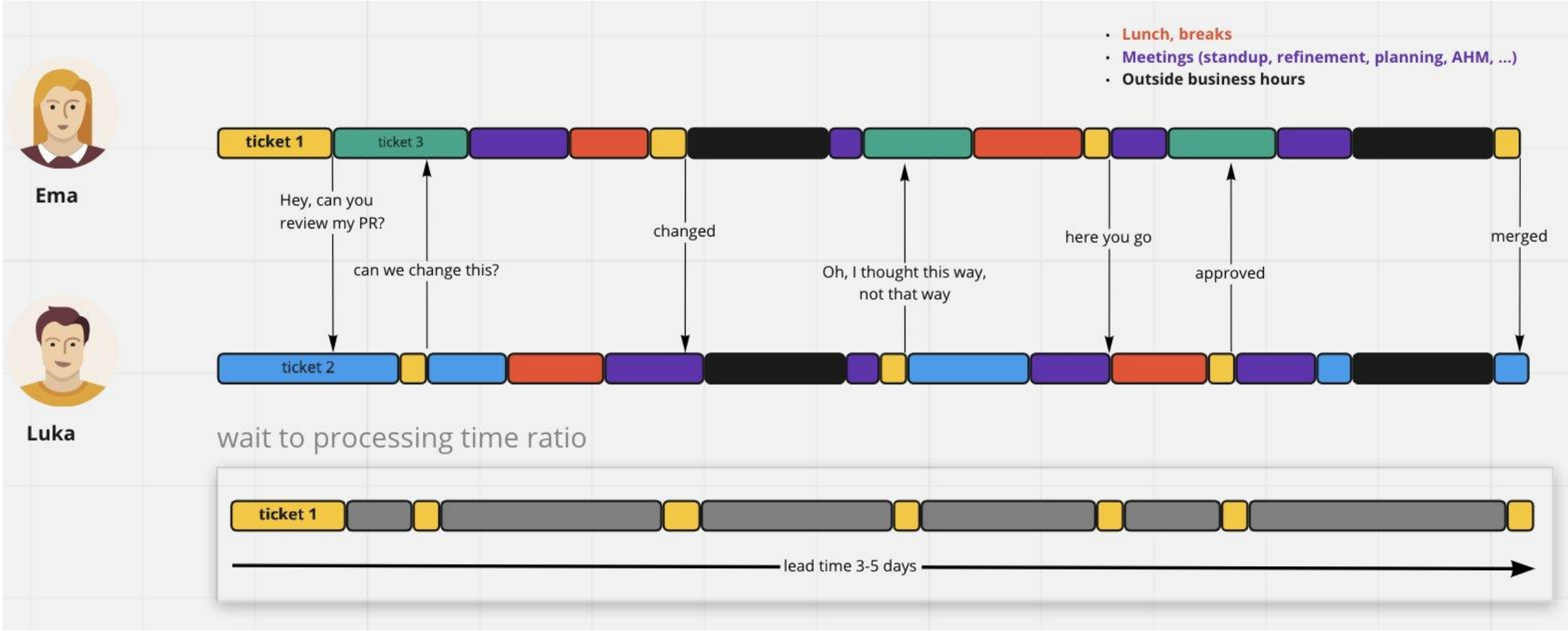
# *Wie funktionieren Code Reviews bei Trunk-Based Development?*

schaltstelle
gemeinsam. selbstständig.

# *Code-Reviews für Feature Branches ist oft ein langer Prozess...*


schaltstelle

# «Aber wir haben keine Zeit für Pair Programming»



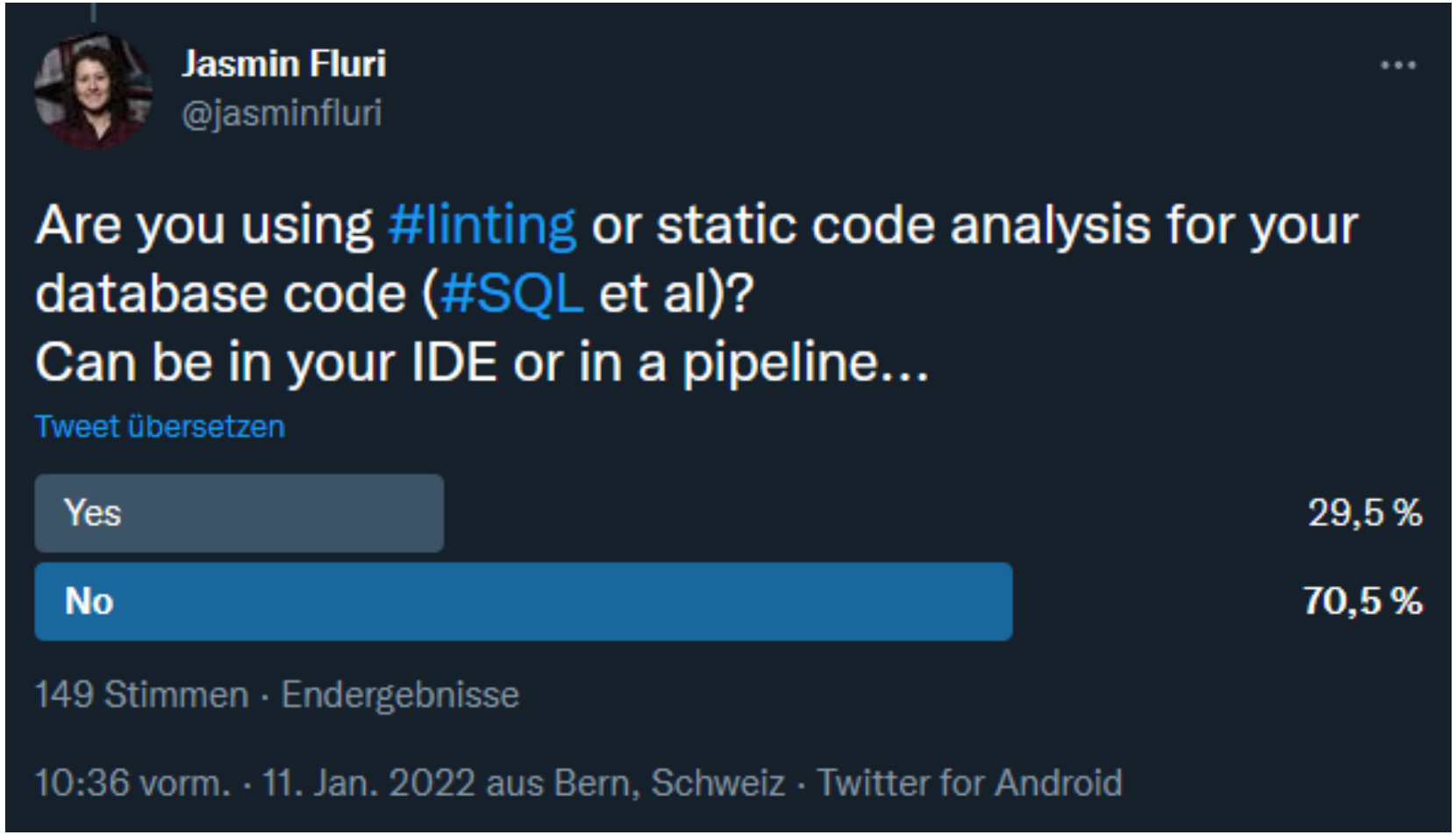

schaltstelle

Source: https://twitter.com/d_stepanovic

# *(2) Statische Code Analyse*

schaltstelle
gemeinsam. selbstständig.

*Um Code-Reviews durchführen zu können, braucht man Standards!*
*Diese Standards sollten automatisch durchgesetzt werden!*

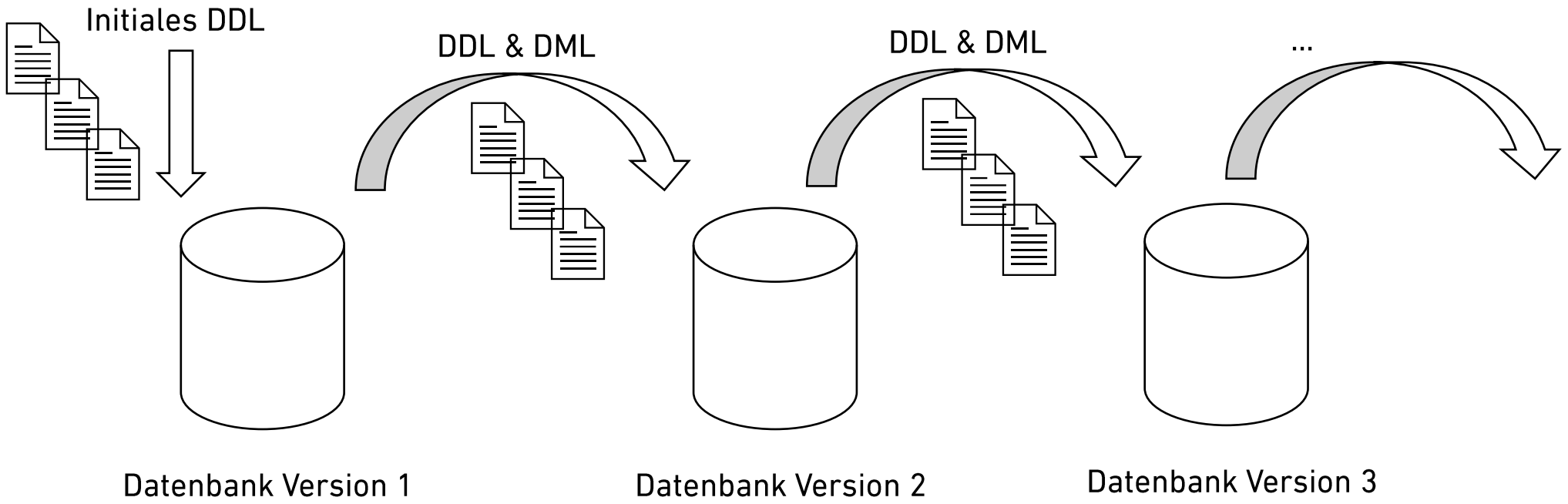



schaltstelle

# *(3) Datenbank Migrationstool*

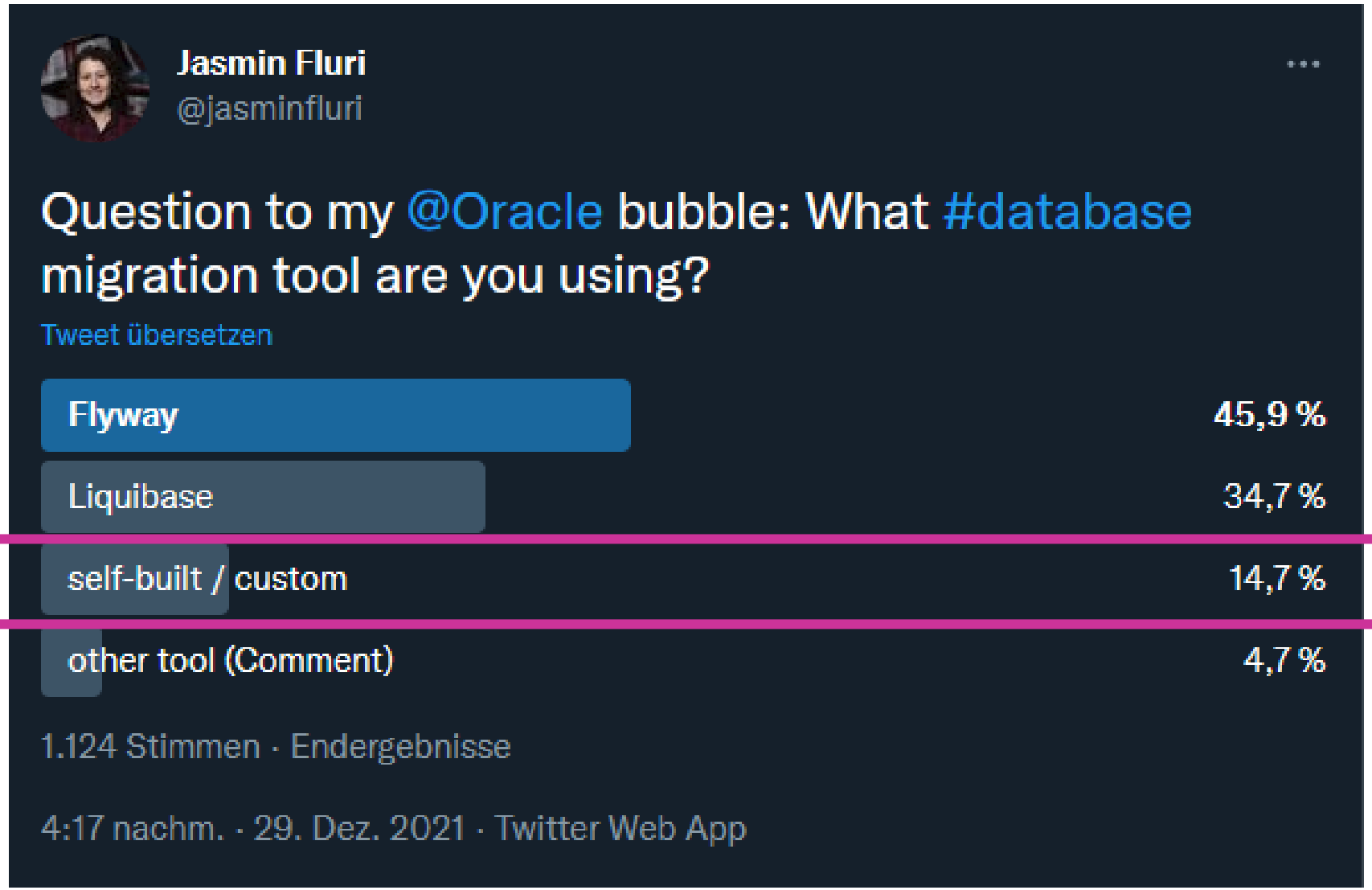
schaltstelle
gemeinsam. selbstständig.

# *Datenbank Schema Evolution*




schaltstelle

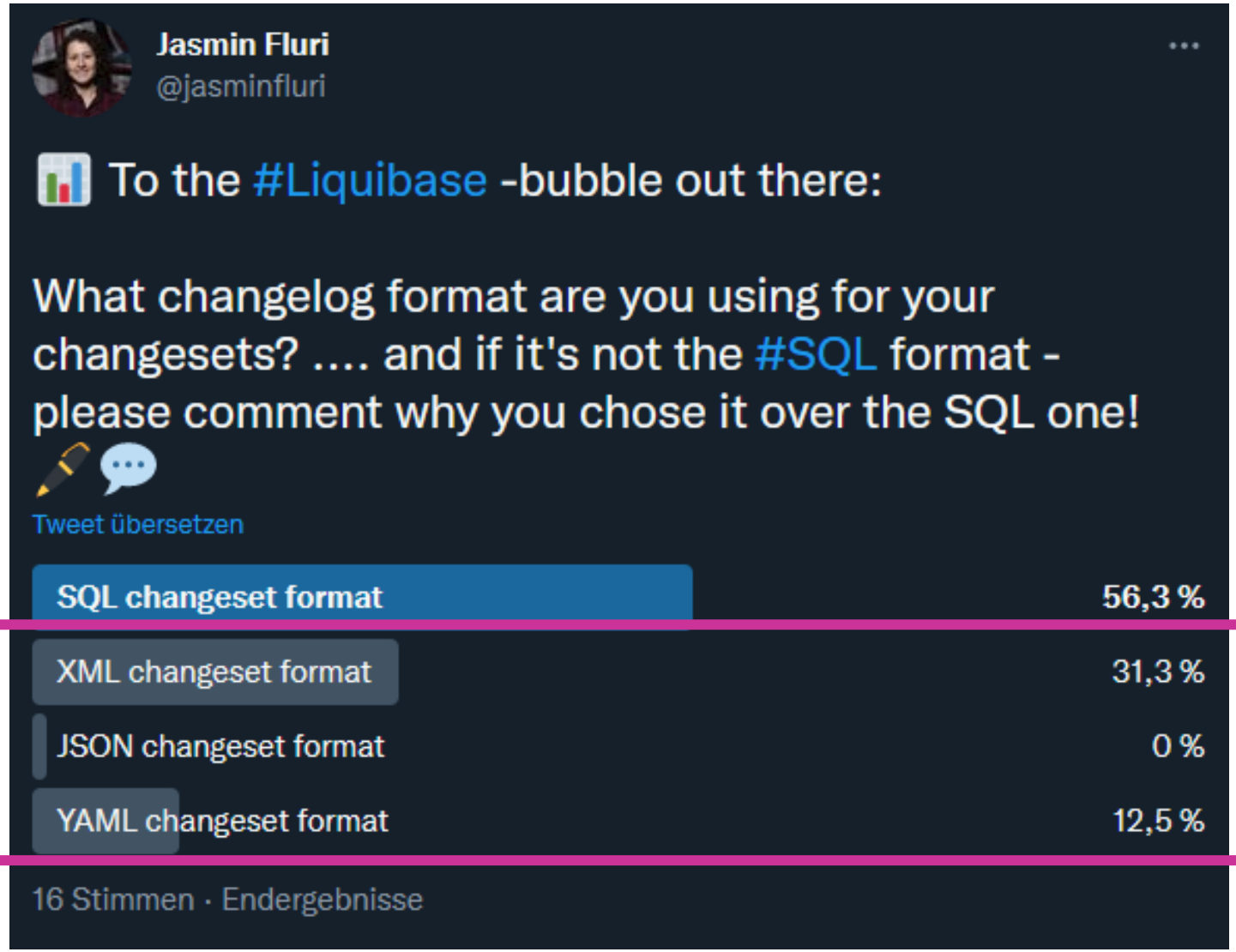
# *Database Migration Tools*




14.7%

schaltstelle

# *Changeset formats – just use SQL!*




43.7%

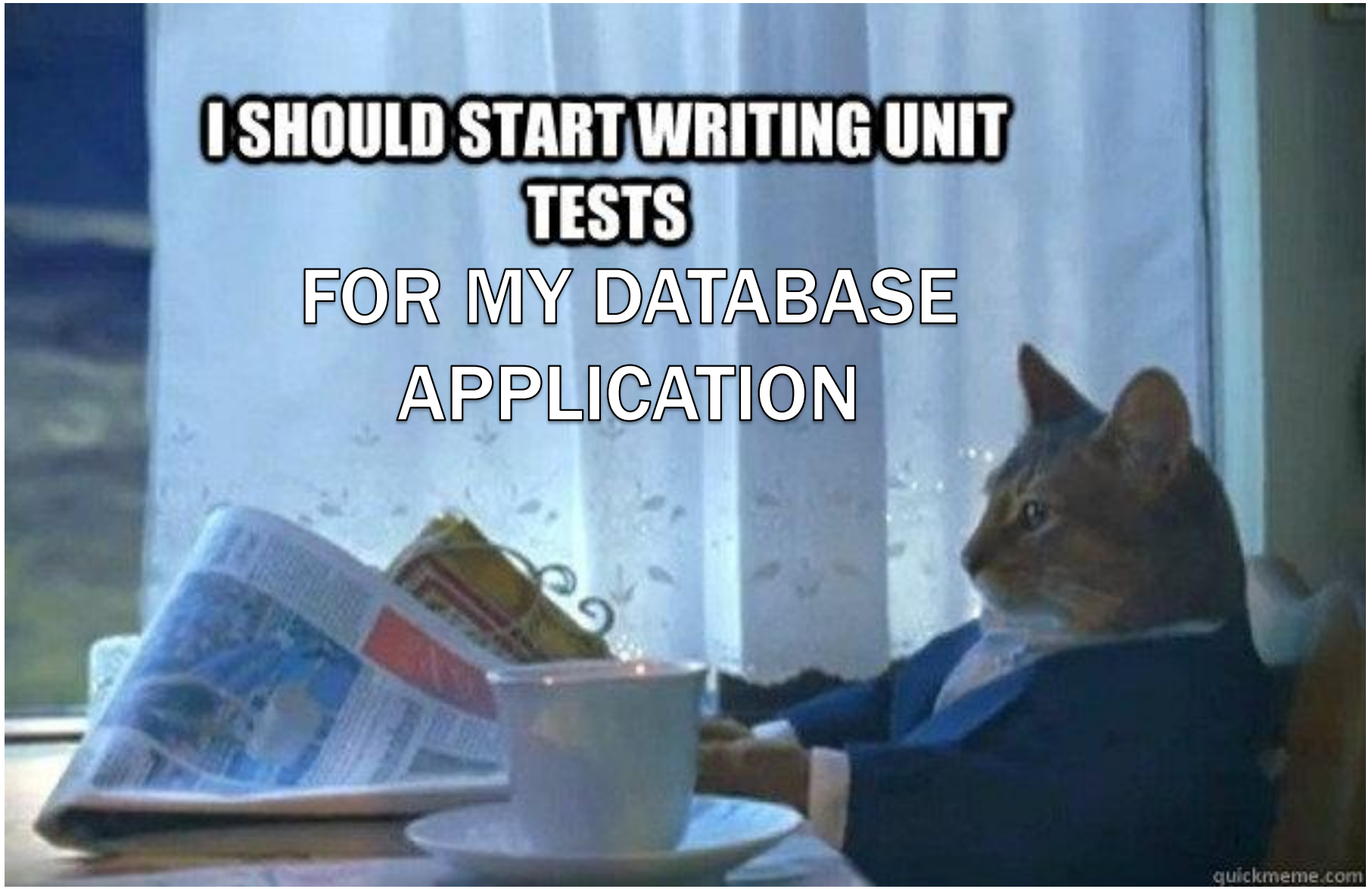
schaltstelle

# *(4) Automatisierte Datenbank Tests*

schaltstelle
gemeinsam. selbstständig.

*Projekte ohne automatisierte Tests wenden etwa 25 % der Kapazität ihres Teams für manuelle Tests auf.*

schaltstelle
gemeinsam. selbstständig.




schaltstelle

# *Testing Tools und Frameworks*

**Unit Testing in der Applikation**

- API Tests
- Integration Tests
- Unit Tests des Backends



**Unit Testing in der Datenbank**

- Unit Tests
- Integration Tests

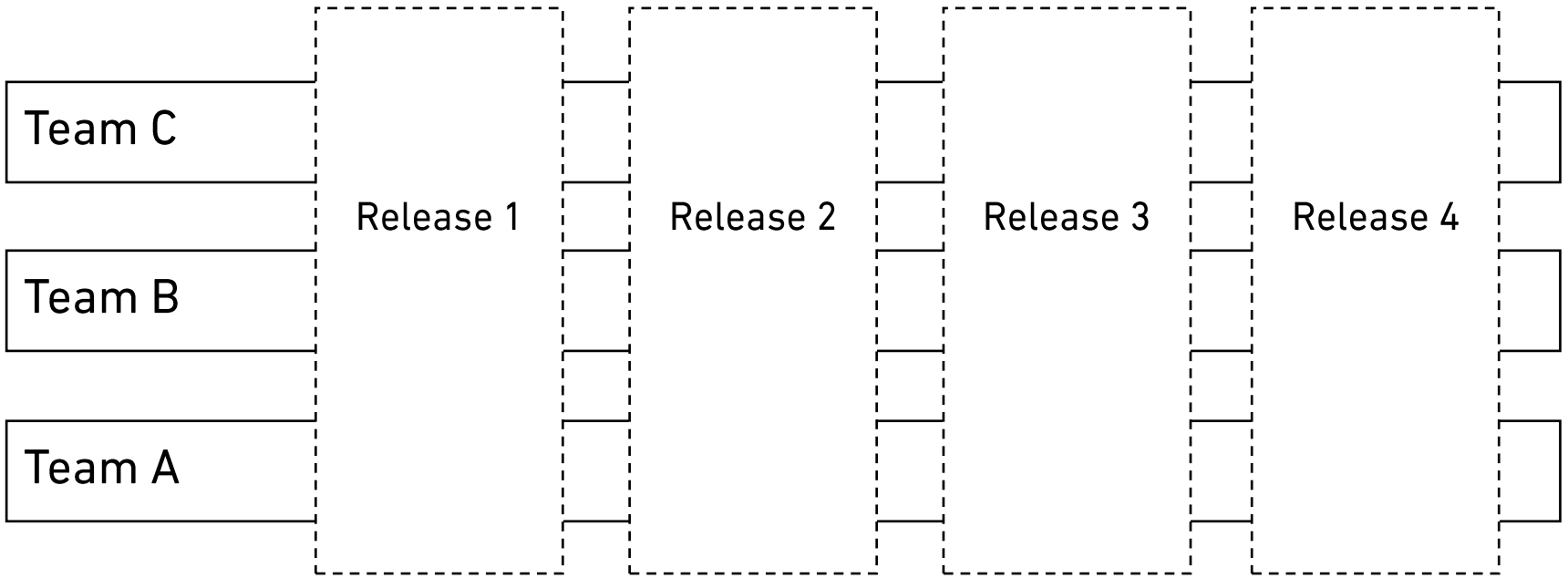


schaltstelle

# *(5) Decoupled Codebase*

schaltstelle
gemeinsam. selbstständig.

# *Was geschieht, wenn die Release-Zyklen an andere Teams (oder Komponenten) gekoppelt sind?*




Dies ist nicht Continuous Delivery!

schaltstelle

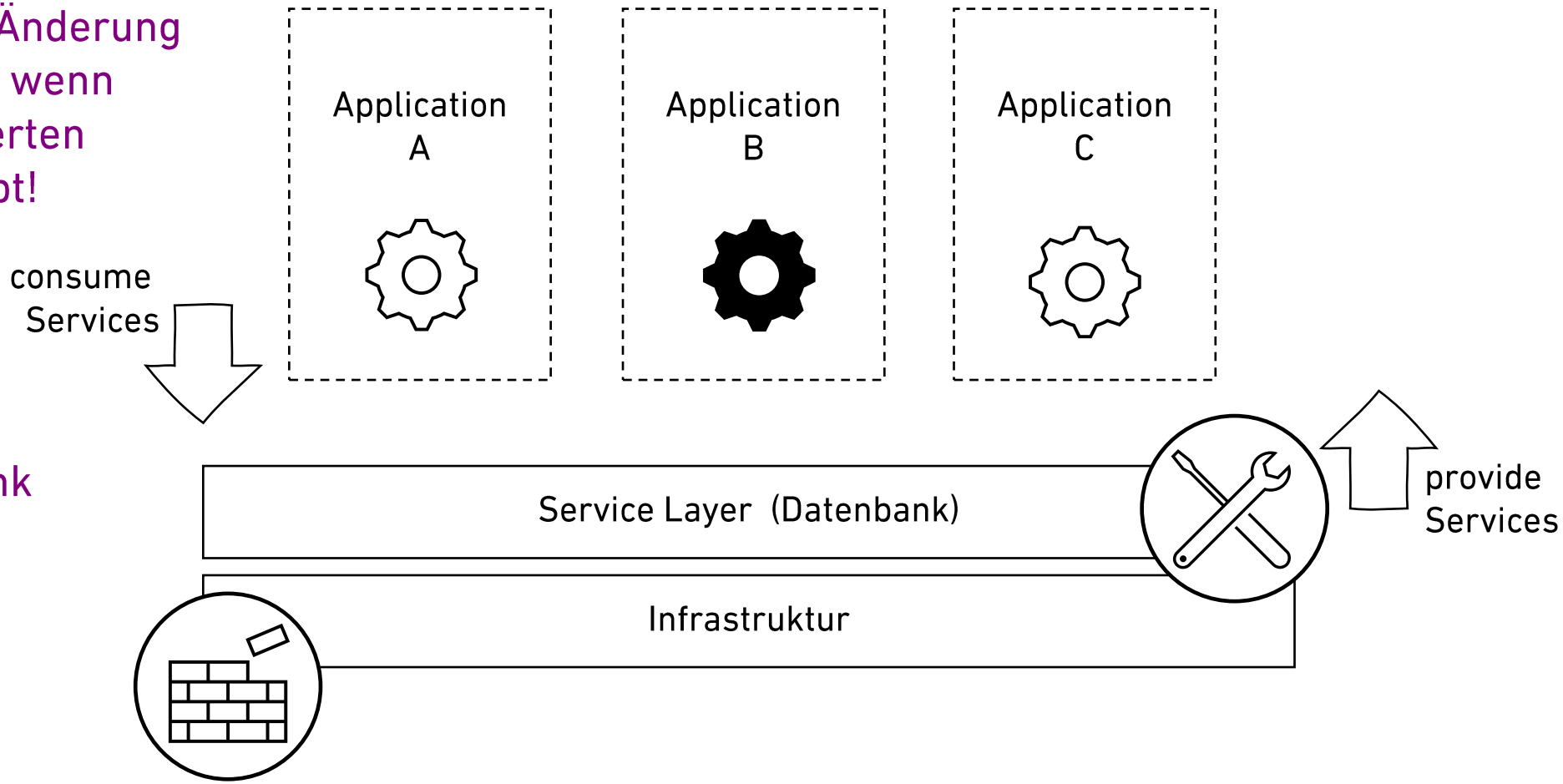
# *Was ist ein Abstraction Layer?*

schaltstelle
gemeinsam. selbstständig.

# *Das Datenbankschema ist ein API – Änderungen daran können für Konsumenten «breaking» sein!*

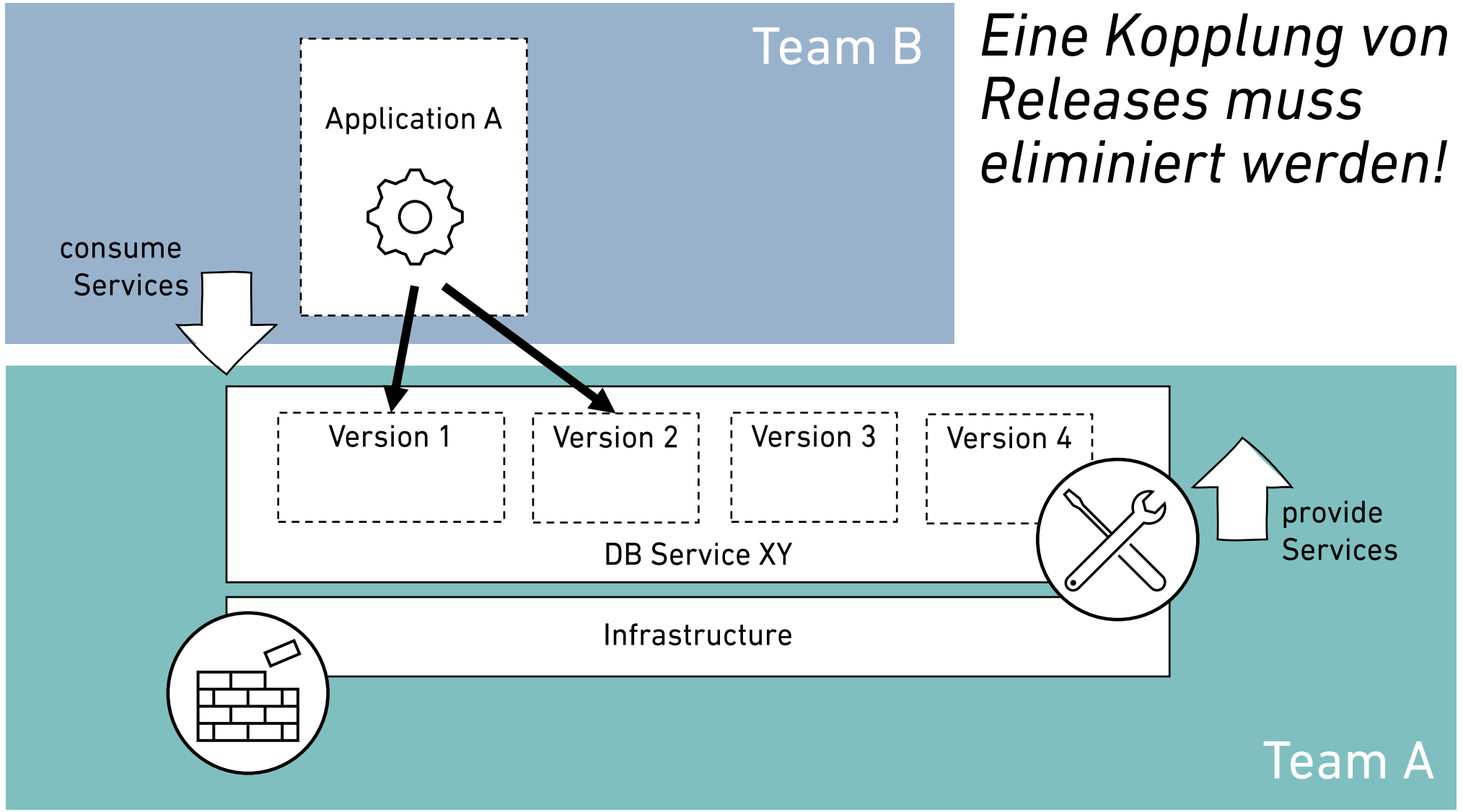
… die Applikationen die Änderung nachziehen müssen, wenn es keinen versionierten Zugriffslayer gibt!

Wenn sich die Datenbank ändert…




schaltstelle

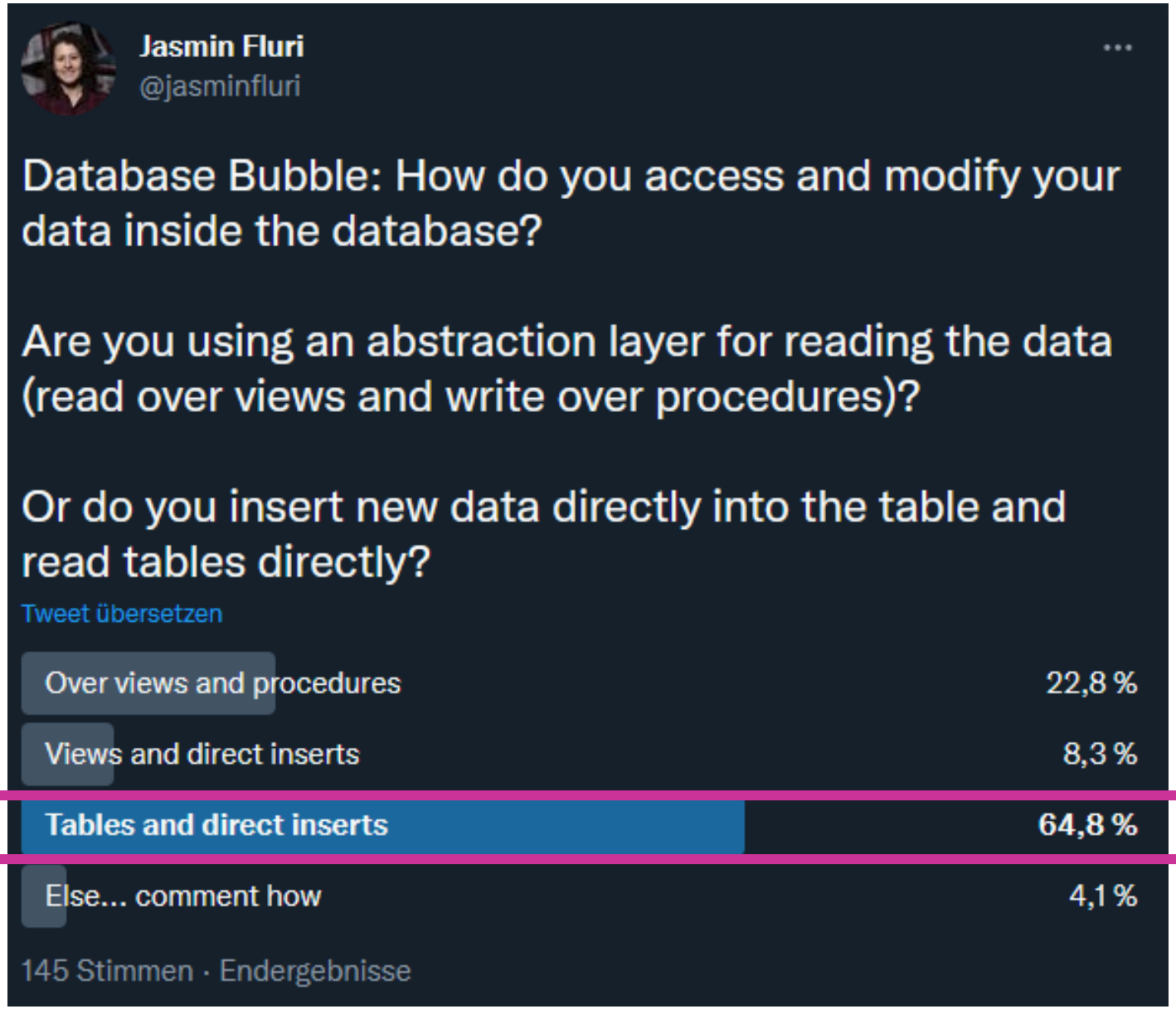
# *Teams wollen unabhängig und kontinuierlich arbeiten!*




*Eine Kopplung von Releases muss eliminiert werden!*

schaltstelle

# … das Problem ist aber leider …

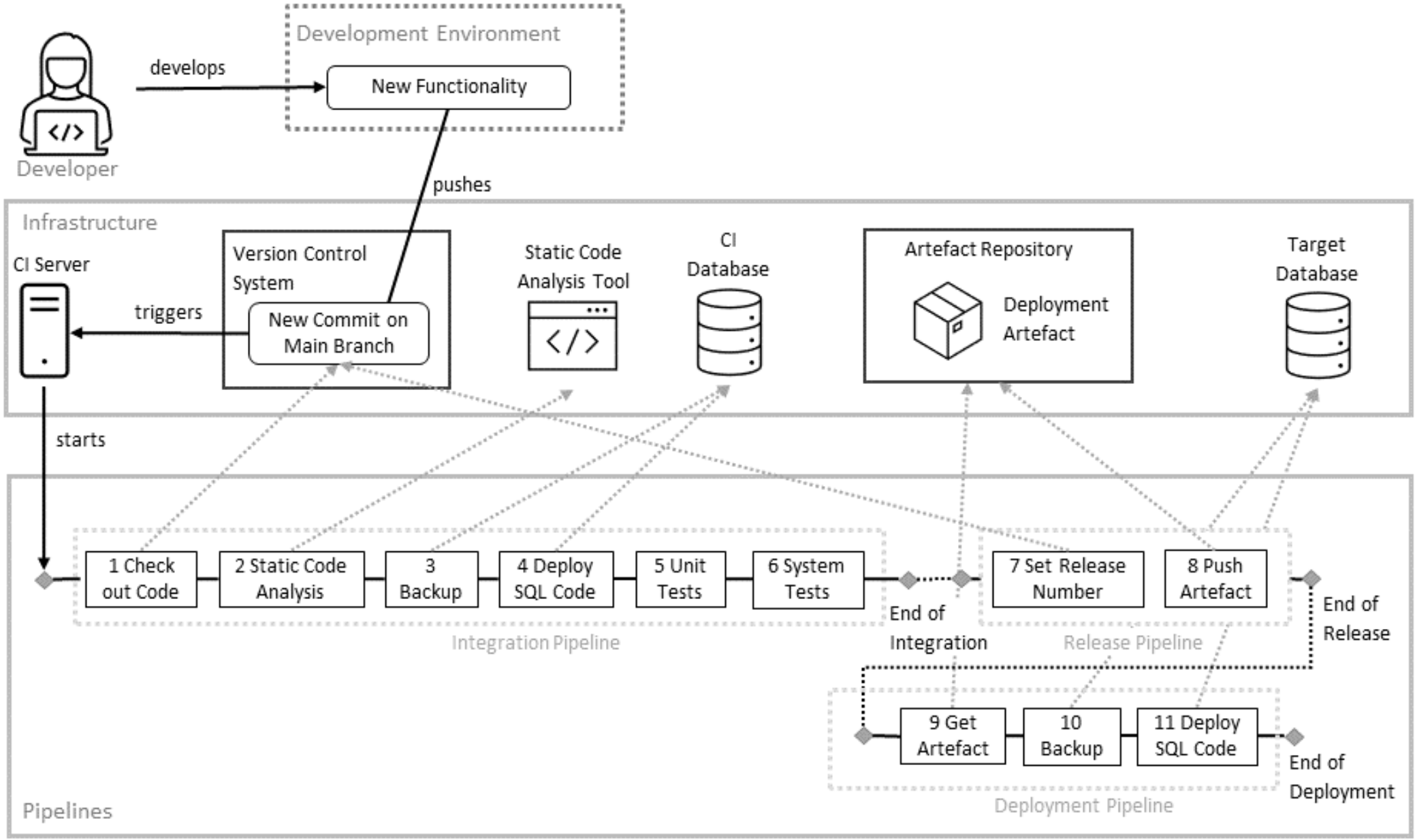



64,8%

schaltstelle

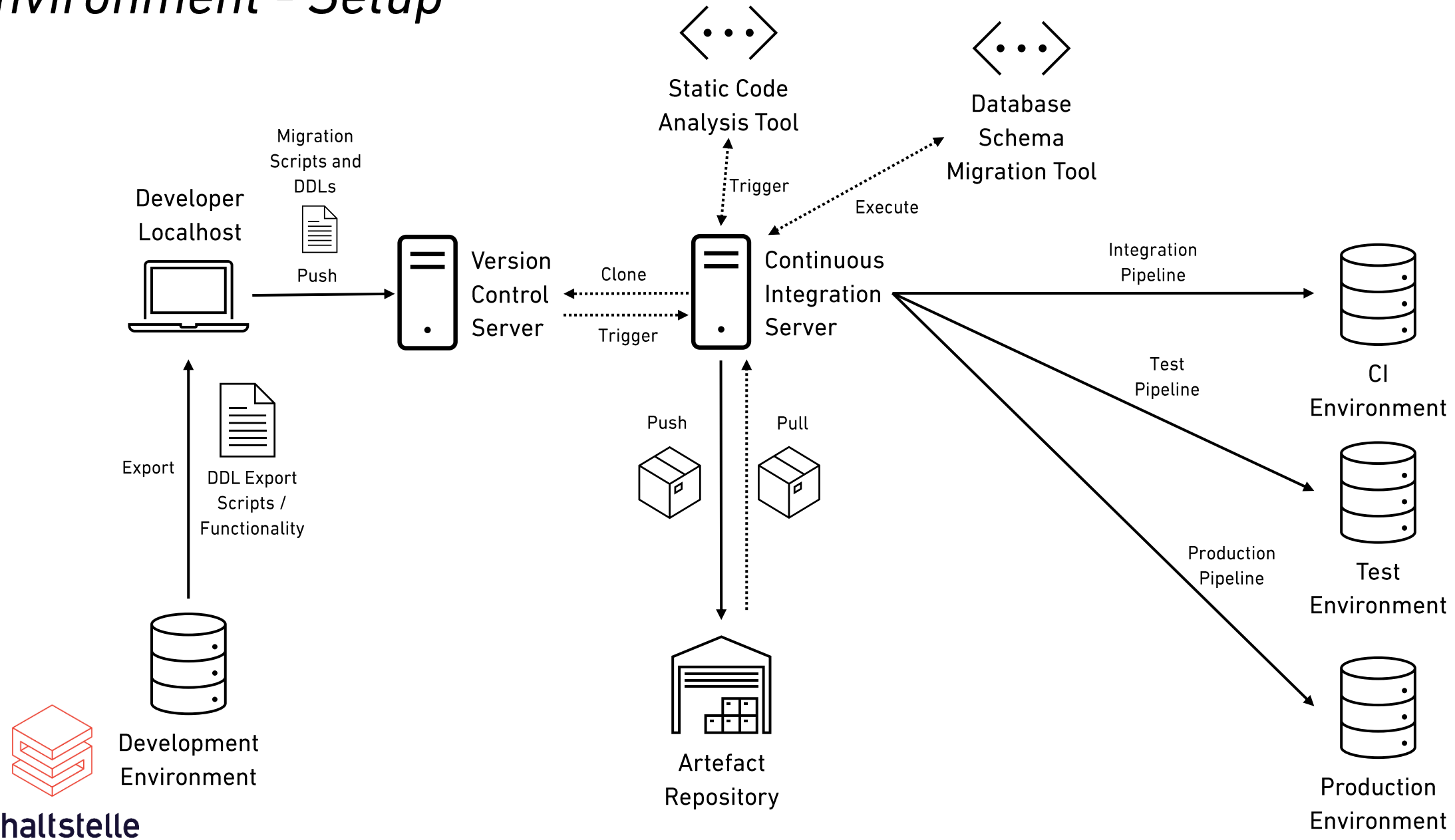
*Jetzt haben wir alle Voraussetzungen, um eine DB CI/CD-Pipeline zu bauen!*

schaltstelle
gemeinsam. selbstständig.




schaltstelle

# Environment - Setup




schaltstelle

# *Was sind die Effekte mit CI/CD?*

schaltstelle
gemeinsam. selbstständig.

*Die Nicht-Automatisierung von Integration und Deployment ist ein Change Preventer!*

schaltstelle
gemeinsam. selbstständig.

*Bei Projekten steigt die Anzahl der Deployments über 5x wenn sie Pipelines haben!*

schaltstelle
gemeinsam. selbstständig.

*Die Deployment Fehlerquote sinkt über 75% wenn Pipelines verwendet werden.*

schaltstelle
gemeinsam. selbstständig.

*Der Cognitive Load der Entwickler sinkt, wenn sie sich auf Pipelines verlassen können!*

schaltstelle
gemeinsam. selbstständig.

schaltstelle
gemeinsam. selbstständig.

*Vielen Dank für eure Zeit!*

*Welche Fragen habt ihr?*

@jasminfluri

@jasminfluri

jasminfluri

# *Ressources / References*

Accelerate Book
https://itrevolution.com/book/accelerate/
DORA 2022
https://cloud.google.com/blog/products/devops-sre/dora-2022-accelerate-state-of-devops-report-now-out
DORA 2021
https://services.google.com/fh/files/misc/state-of-devops-2021.pdf
The best path to long term success is slow :
https://www.nytimes.com/2017/07/31/your-money/the-best-path-to-long-term-change-is-slow-simple-and-boring.html
Martin Fowler : Continuous Integration
https://www.martinfowler.com/articles/continuousIntegration.htm
Stonebreaker – Database Decay
http://people.csail.mit.edu/dongdeng/papers/bigdata2016-decay.pdf
Dragan Stepanovic – Twitter
https://twitter.com/d_stepanovic

schaltstelle