Managing API Evolution in Microservice Architecture

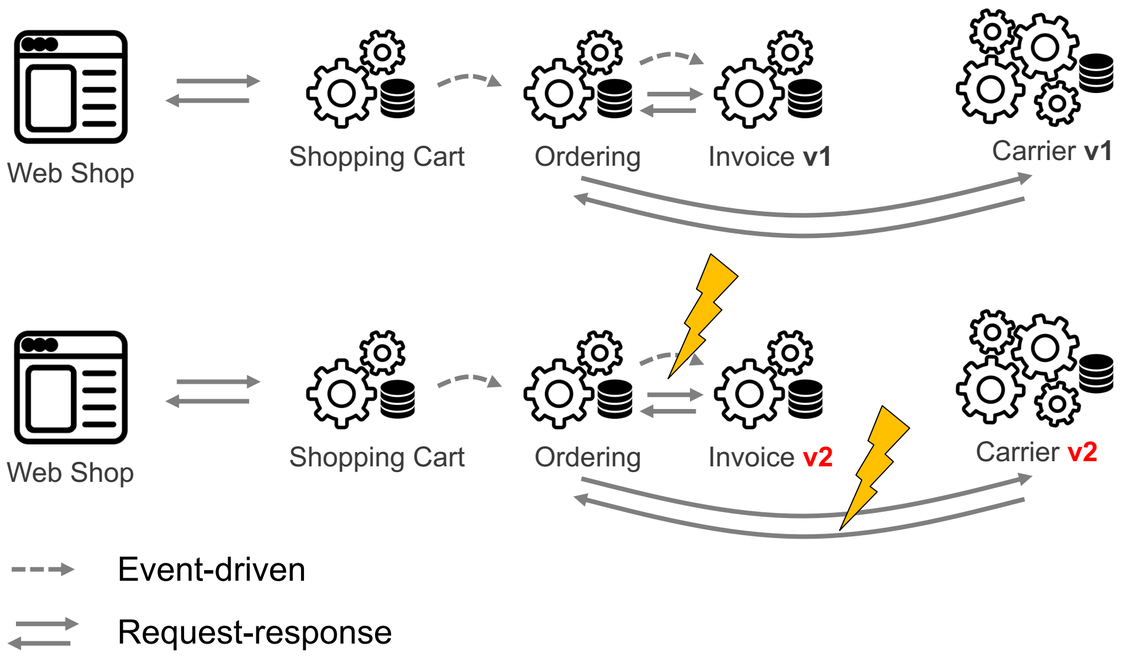
Alexander Lercher
Software Engineering Research Group, University of Klagenfurt

UNIVERSITÄT KLAGENFURT

Contact

Motivation

- **Loosely coupled (micro-)services** provide **no immediate feedback for consumers on API changes**
- **Consumer services may break** or exhibit unexpected behavior after calling the changed API
- Affected development teams rely on **active communication and collaboration** with provider teams



Changelog for Invoice **v2**

Shipping address behavior changed:
- mandatory
- optional, `null` means shipping address is invoice address

Changelog for Carrier **v2**

Shipping information endpoint renamed:
- `GET /orders/{orderId}`
- `GET /orders/{orderId}/shipping`

Expected delivery date format changed:
- `YYYYMMDD`
- `YYYY-MM-DD`

Research Questions

RQ1 Which **strategies** do developers follow to **introduce and communicate API changes** in loosely coupled systems, and what **challenges** do they face?

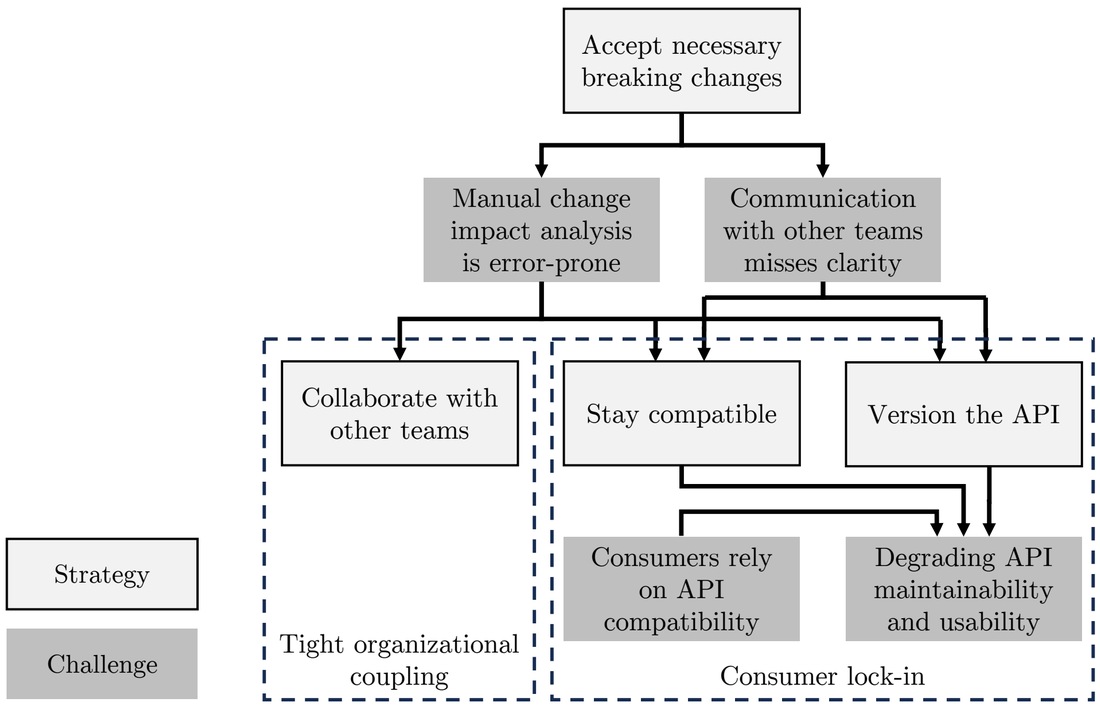
RQ2 How can **structural and behavioral changes of REST APIs** be automatically **extracted** based on the source code?

RQ3 How can relevant REST API changes be automatically **communicated to consumer teams** affected by these changes?

Catalogue of API Evolution Strategies and Challenges (RQ1)

- **Semi-structured interviews** with **n=17** developers and architects from 11 companies
- **Open coding** used in grounded theory

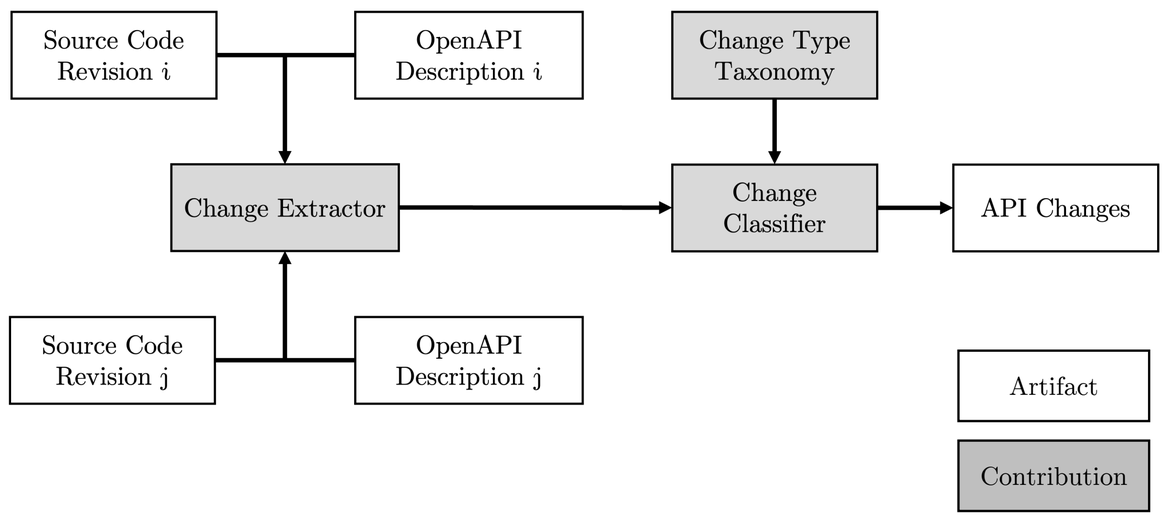
- Evolution strategies focus on **backward compatibility and close collaboration**
- Evolution challenges comprise **manual change analysis and unclear communication**
- Close collaboration creates the problem of **tight organizational coupling**
- Backward compatibility and unclear communication result in **consumer lock-in**



Alexander Lercher, Johann Glock, Christian Macho, and Martin Pinzger. 2023. Microservice API Evolution in Practice: A Study on Strategies and Challenges. arXiv:2311.08175 [cs.SE]

REST API Change Extraction from Source Code Changes (RQ2)

- **Change Extractor** detects structural changes in OpenAPI specification and behavioral changes in source code
- An extended **Change Type Taxonomy** comprises structural and behavioral REST API changes
- The **Change Classifier** translates low-level changes to the structural and behavioral REST API changes



Communication of REST API Changes to Affected Teams (RQ3)

- **API providers publish REST API changes** extracted from the source code repository
- **API consumers register their dependencies** via a web interface or configuration file in their repository
- Consumers receive **targeted change notifications** with multiple levels of detail and timeliness
- User study to **evaluate the timeliness, completeness, and understandability** of the change notifications