GI FG TAV
Bewertung von Software-Architekturen

# Architektur-Reviews für nachhaltige IT-Landschaften

Joachim Nelz, Software Architekt & Team Lead, S&N Invent

03.07.2025

S&N INVENT Value Driven Performance.

Lufthansa Cargo

# WER WIR SIND

Wir sind Technologen, Architekten, Entwickler und Fachexperten – immer kundenorientiert, neugierig und pragmatisch

Mit modernen Frontend- und Backend-Technologien schaffen wir professionelle Enterprise- sowie Cloud-Lösungen und betreuen diese über den gesamten Lifecycle.

Unsere Expertise bringen wir in innovative Softwareprojekte mit unseren Kunden und in unsere Produkte ein.

1991

>30 Jahre erfolgreiche Projektarbeit und Beratung

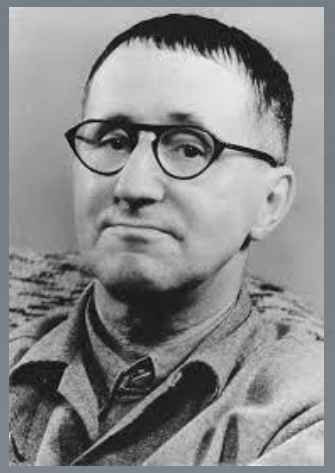

>500 Mitarbeiter*innen

85 Mio. EUR Umsatz

# Inhalt

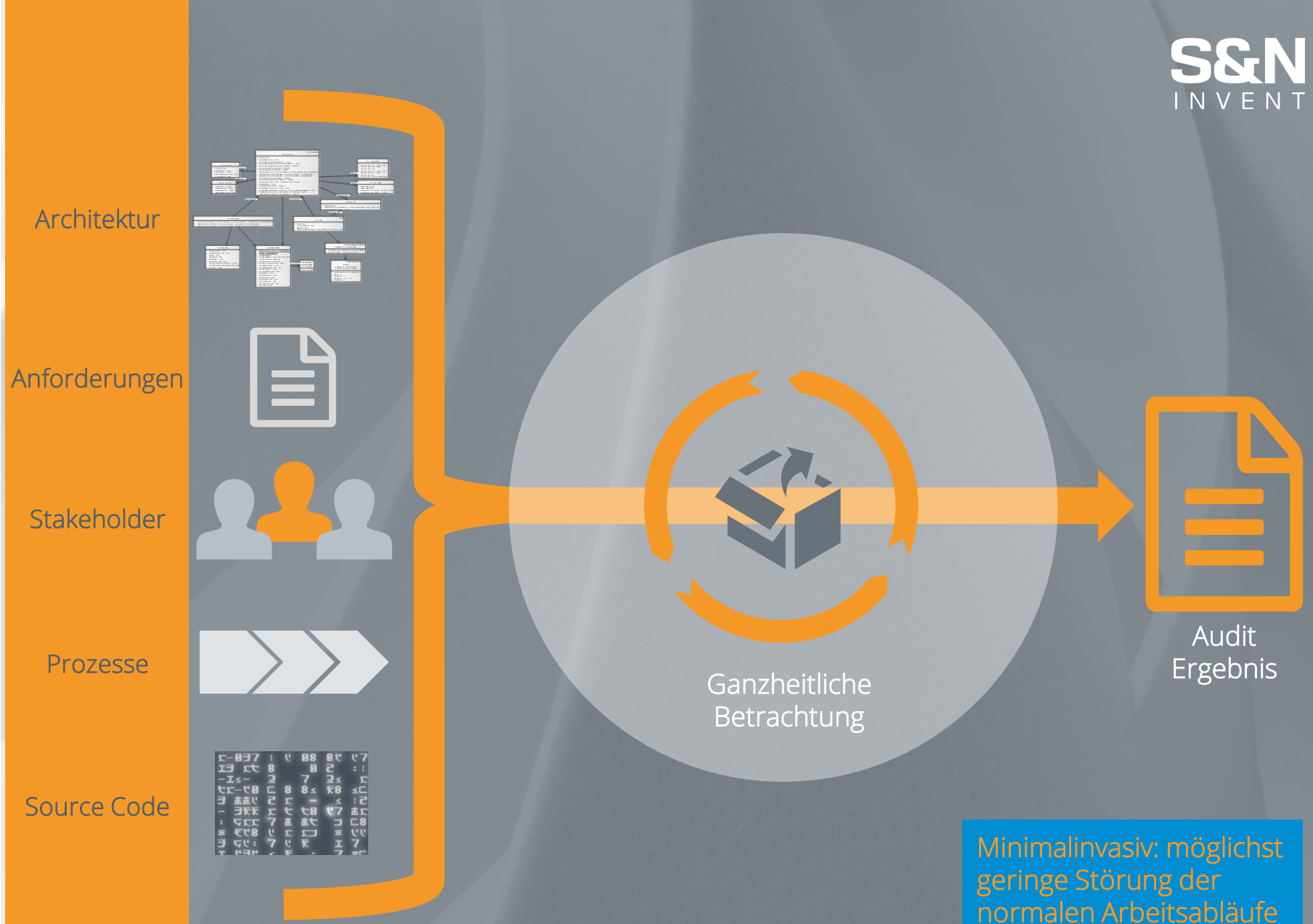
1. Ansatz & Vorgehen
2. Praxis Beispiele
3. Best Practices & Antipatterns

# Geschichten von Herrn K: Rechtsprechung



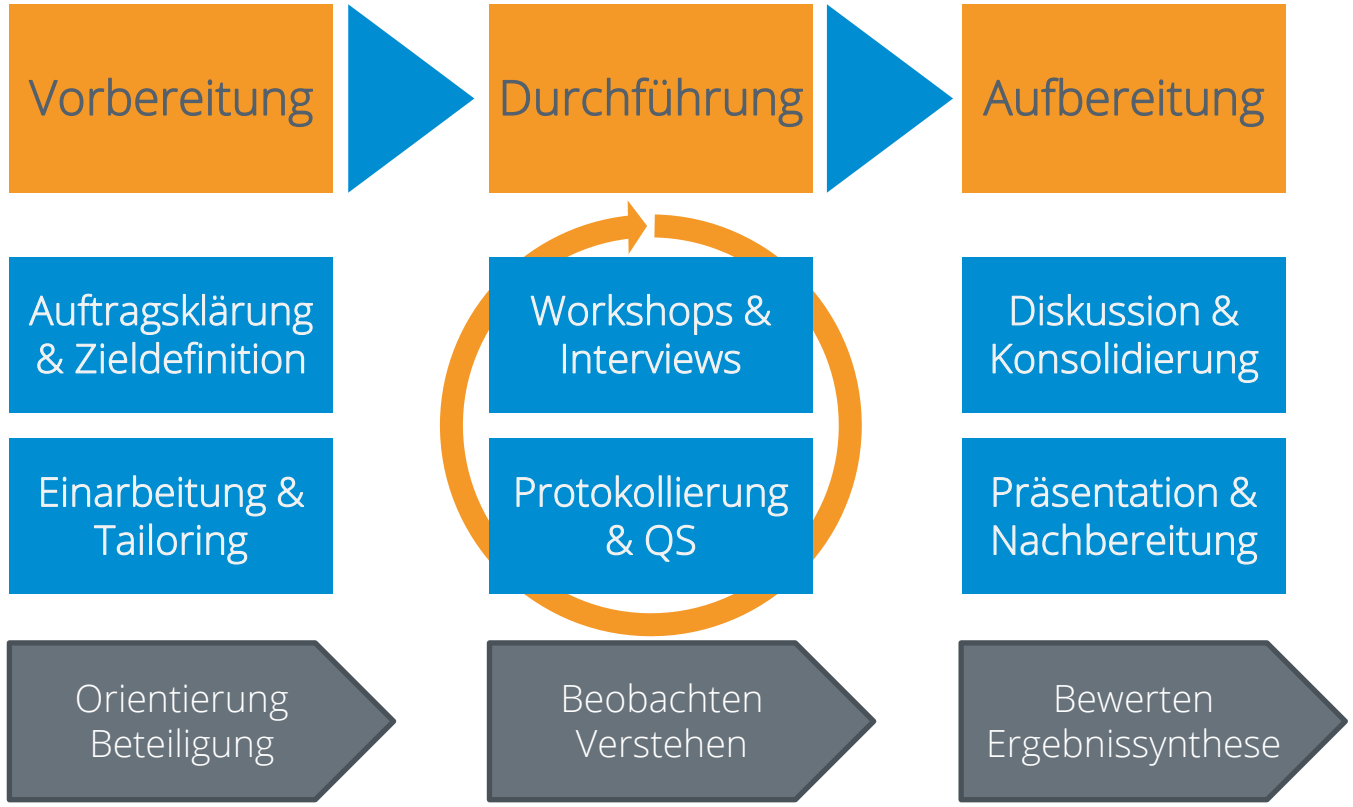
- Herr K. nannte oft als in gewisser Weise vorbildlich eine Rechtsvorschrift des alten China, nach der für große Prozesse die Richter aus entfernten Provinzen herbeigeholt wurden.
- Auch kannten diese herbeigeholten Richter die Gebräuche und Zustände der Gegend nicht aus der alltäglichen Erfahrung.
- Unrecht gewinnt oft Rechtscharakter einfach dadurch, dass es häufig vorkommt.
- Die Neuen mussten sich alles neu berichten lassen, wodurch sie das Auffällige daran wahrnahmen.

# Ganzheitliche Betrachtung aller relevanten Perspektiven



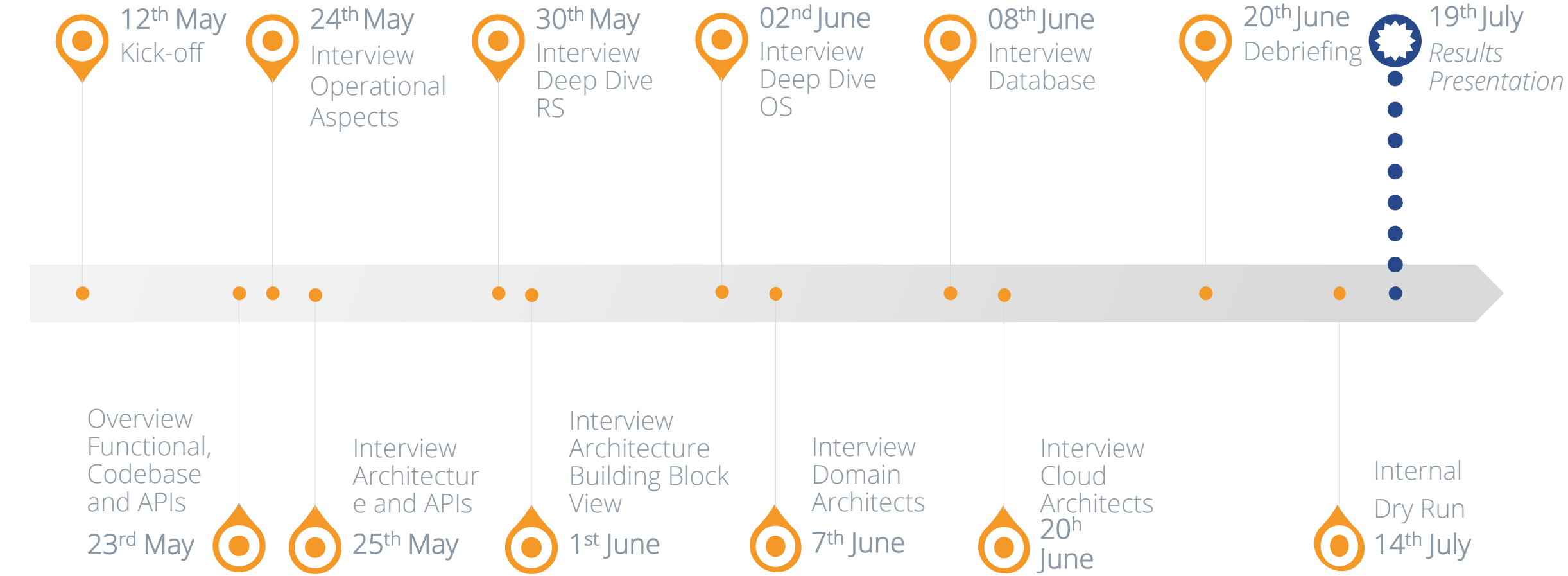

Minimalinvasiv: möglichst geringe Störung der normalen Arbeitsabläufe durch das Review.

# Das S&N Review Referenz-modell




- ✓ Interviews ermöglichen schnellen Know-How Transfer
- ✓ Regelmäßige Abstimmung schafft Transparenz und erlaubt Anpassungen
- ✓ Hinreichende Analysetiefe trotz kompakter Timebox

# Timeline: Review of large Legacy System




Phase 1 | Phase 2 | Phase 3


Lufthansa Cargo

# Software Architektur-Review
# Praxis Beispiele

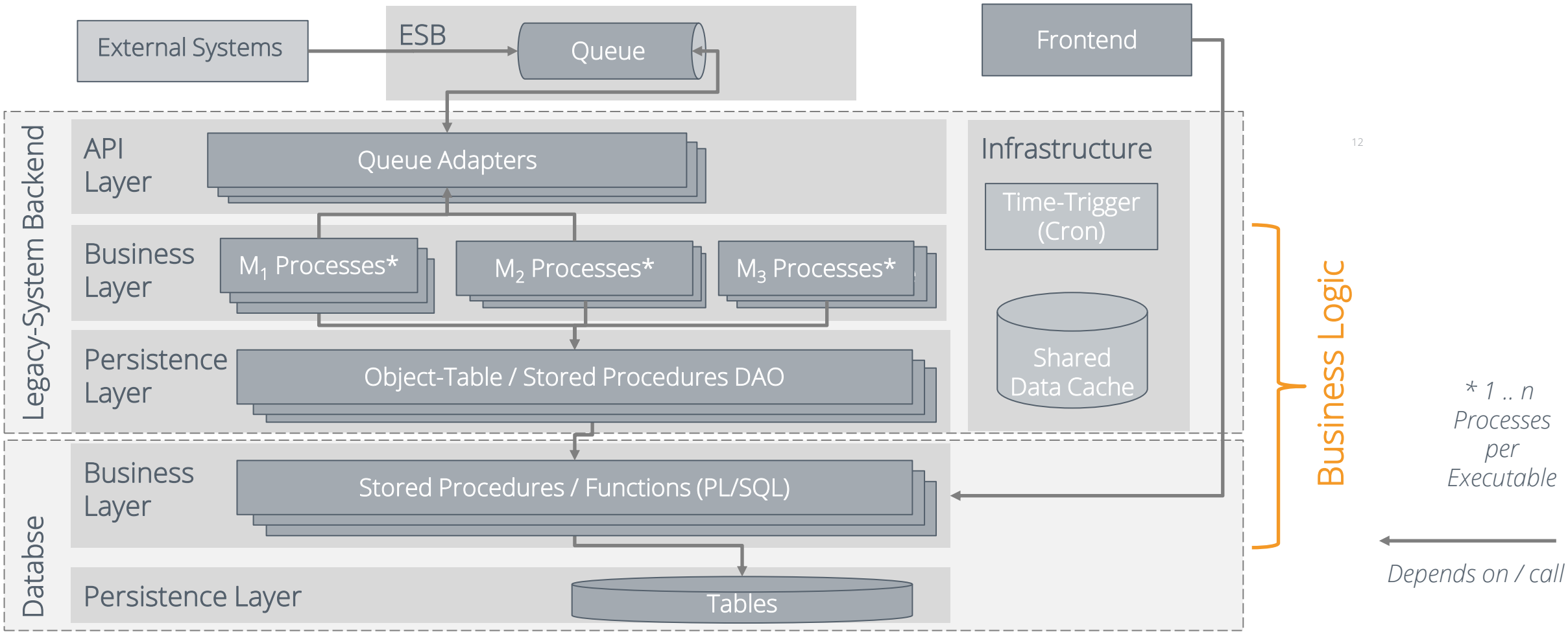
# Großes Legacy System

- Großes Legacy System, integriert mit zahlreichen nachgelagerten Backend Systemen

- Review Auftrag:
  - Nachhaltigkeit der Architektur
  - Stabilität & Performance
  - Wartbarkeit, Anpassbarkeit und Cloud Migration

- Tech-Stack:
  - PERL Frontend
  - Zahlreiche C/C++ Backend Prozesse
  - Oracle DB mit Stored Procedures, ESB via MQ

# Current Architecture – Layer View

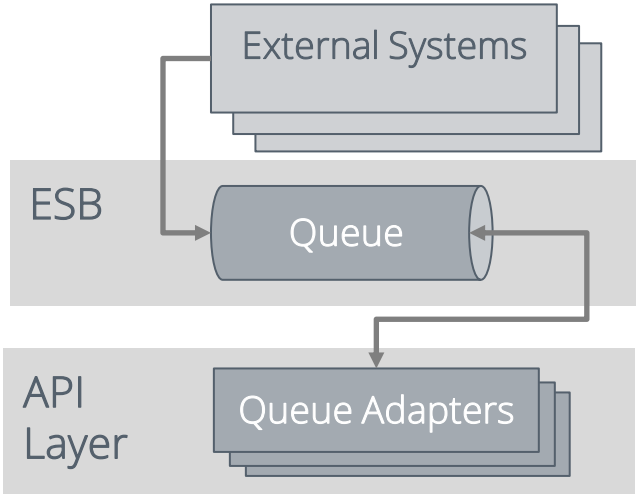



**Assessment**

✗ Limited adaptability due to risk of potential side effects

# Architecture: Communication / API

- Asynchronous communication by inbound and outbound messages
  - Data load is temporarily cached in message queues, with guaranteed message delivery
  - Messages have Time-To-Live – which avoids queue overflow
- Reduced API consumer dependency with tolerant reader pattern
- No technical message validation
- API behaviour not completely documented (happy case only)




Assessment

✓ Resilient and performant asynchronous communication style

✓ Legacy-System temporal decoupled from external systems

✘ Risk for undetected error cases and data inconsistency due to missing message validation

# Conclusion

Messaging interfaces contribute
to the performance and resilience of Legacy-System

Current architecture has many built-in performance optimizations
Further increase would require extensive redesign

Adaptability limited to small changes only,
due to low modularity and missing tests and documentation
Complete separation of main modules unfeasible

Cloud migration feasible only per Lift & Shift.
Thus, no benefits of Cloud native (dynamic scaling, redundancy).

# Nach dem Review

- Lift & Shift Migration:
  - Motiviert durch die Stilllegung des bisherigen Rechenzentrums.
- Hohe Anforderungen an Änderbarkeit und Performance:
  - Neue Anforderungen erfordern eine hohe Anpassbarkeit des Systems.
  - Fortbestehender Anspruch an sehr hohe System-Performance.
- Ende der weiteren Entwicklungspotentials des aktuellen Systems
  - Zahlreiche Performance-Optimierungen implementiert, jedoch ausgeschöpft.
  - Geringe Modularität begrenzt bisher größere funktionale Anpassungen.
- Initiierung einer Designstudie (Start Q3/2024) basierend auf Review
  - Entwicklung einer neuen zukunftsfähigen Architektur
  - Definition des zukünftigen Systemaufbaus mit Fokus auf Modularisierung.
  - Ziele: Erfüllung der bestehenden Performance-Anforderungen und Unterstützung umfangreicher fachlicher Weiterentwicklungen.

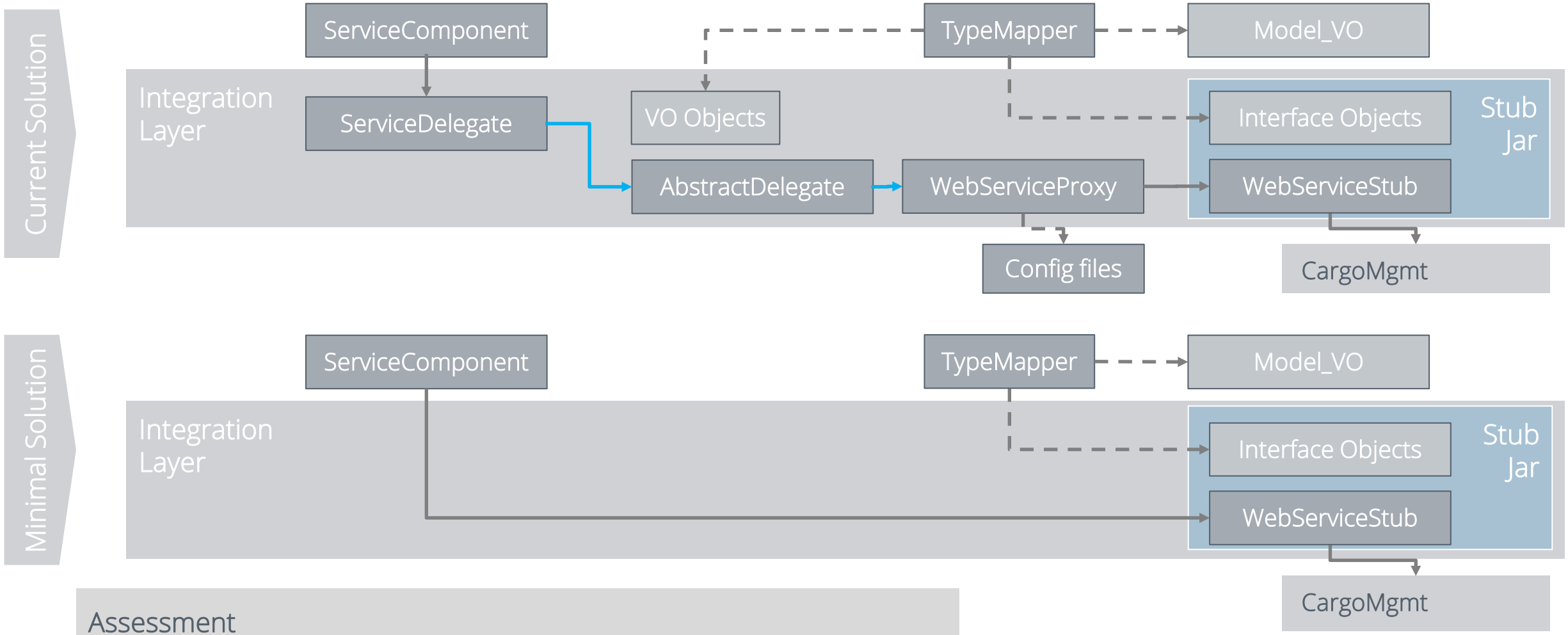
# Digital Touch Point

- Web Frontend des zentralen Produktionssystems, für LH Cargo Kunden als Alternative zur B2B API

- Review Auftrag:
  - Bewertung der Qualität von Architektur und Codebase hinsichtlich technischer Schuld
  - Wartbarkeit und Anpassbarkeit
  - Nutzbarkeit der Dokumentation
  - Backend-Integration

- Tech-Stack:
  - Java, Liferay Portal

# Deep Dive Service Integration vs Minimal Solution

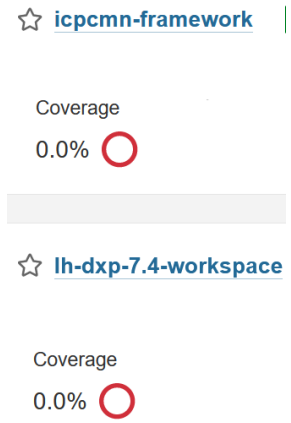



Assessment

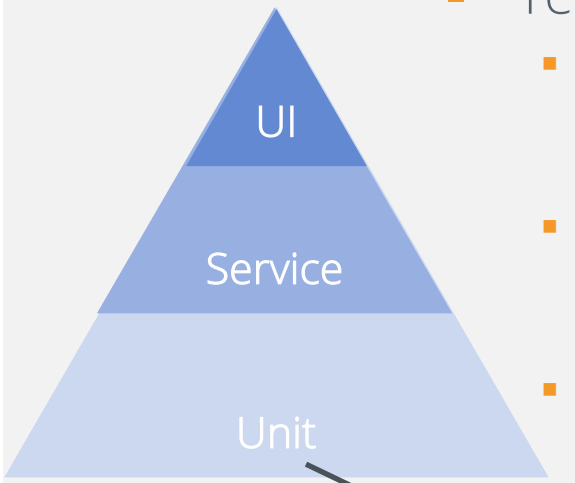
- ✘ Significant overhead of current integration without benefit

# Zero Test Coverage

- No test coverage within code base
  - No Unit Tests
  - No Integration Tests




- This contradicts established Best Practices →




- Test Pyramid by Mike Cohn
  - Few UI Tests
    - Mostly manual, cause high effort
    - Hard to maintain
  - More Service-Level Tests
    - Integration Tests, run „headless“ without UI
    - More easily automated than UI Tests
  - High coverage by Unit Tests
    - Automated and continuously used

Early and specific feedback

## Assessment

- ✘ Completely missing test coverage is not state of the art
- ✘ Maintenance and adaption is impeded due to risk of unwanted side-effects on each change
- ✘ Refactoring is risky without safety net, which in turn leads to accumulation of technical debt in the future.

# Flawed Exception Handling

- Exceptions are caught, logged and then ignored
  - Often w/ wrong log level
  - Often w/o stacktrace recording

```
try {
 localDimHeightVO = PortletUtil.convertToOtherUnit(UnitConversionVO.KEY_DIM, height, "I", "C", 1);
 localDimLengthVO = PortletUtil.convertToOtherUnit(UnitConversionVO.KEY_DIM, length, "I", "C", 1);
 localDimWidthVO = PortletUtil.convertToOtherUnit(UnitConversionVO.KEY_DIM, width, "I", "C", 1);
} catch (BusinessException e) {
    [redacted]PersistantMapper.LOG.info(" unit convertion error  " + e.getMessage());
}
if(localDimHeightVO!=null && Double.compare(localDimHeightVO.getUnitValue(), 0.0)!= 0 ){ …
```

Null checks for further error handling

- Throwing generic Exception instead of dedicated one

Assessment

- ✘ Exception Handling is flawed, which impedes maintenance and adaptability
- ✘ This Anti-Pattern is widespread over the complete codebase
- ✘ Logging often with wrong log level, which convolutes log files and makes errors harder to track

# Nach dem Review

- Projektstart zur Portlet-Ablösung
  - Vollständige Neuentwicklung der kritischsten Portlets durch einen neuen Partner.
- Qualitätssicherung
  - Implementierung gezielter Maßnahmen zur Steigerung der Testabdeckung und zur Korrektur des Exception-Handlings.
- Architektur Ansatz
  - Verwendung von ESB-Schnittstellen für sämtliche Integrationen zur konsequenten Kapselung zwischen Web-Portal und Produktionssystem.
  - Ausrichtung an den Lufthansa Cargo Vorgaben für Schnittstellen-Transparenz und Entkopplung innerhalb der Applikationslandschaft.

# Greenfield Integration

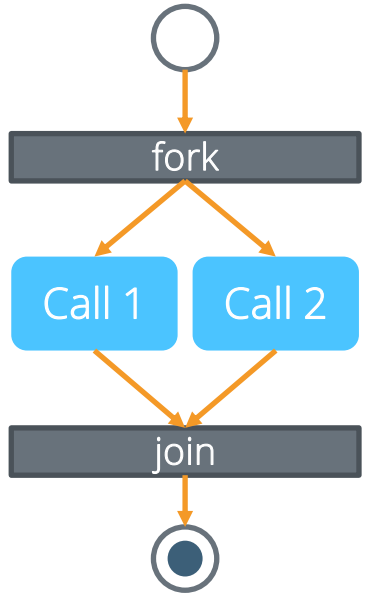
- Neu entwickeltes globales Buchungs- und Reservierungssystem, bestehend aus Micro-Services, die zahlreiche nachgelagerte Backend Services integrieren

- Review-Auftrag:
  - Validierung Tragfähigkeit der Architektur
  - Qualitätssicherung der Implementierung
  - Entwicklungsprozess, CI/CD Pipelines

- Tech-Stack: Java/Spring Boot

# Service Decoupling

- External Service adapters implemented using Spring Services
- Resilience Patterns: TimeLimiter and CircuitBreaker
  - Fallback method returns basic errors, no retries (in code)
  - Each timeout configured individually
- Synchronous service calls are parallelized by CompletableFuture when possible




Assessment

✓ Resilience Patterns are used, such as CircuitBreaker

✓ Timeouts are configured and discussed with teams

➢ Parallelization of service calls improve performance, but induces complexity due to multi threading

➢ No real fallback used, caching might be an option

# Backend Best Practices revisited

- Resilience Patterns on database access
  - Retry on Deadlock

```
@Transactional
@Retry(name = "throwingDeadlockException")
public void createBooking(BookingResponseVO responseVO) {
    […]
    bookingsServiceRepository.save(dbBooking);
}
```

- Logging and Monitoring with Elastic-Stack
  - Application Performance Management (Network requests, Call tree, Database-Queries)
  - Log-Gathering and Analysis-Frontend

- Azure SQL Database Monitoring
  - Index-Improvements proposed by Azure, reviewed ~ once a month

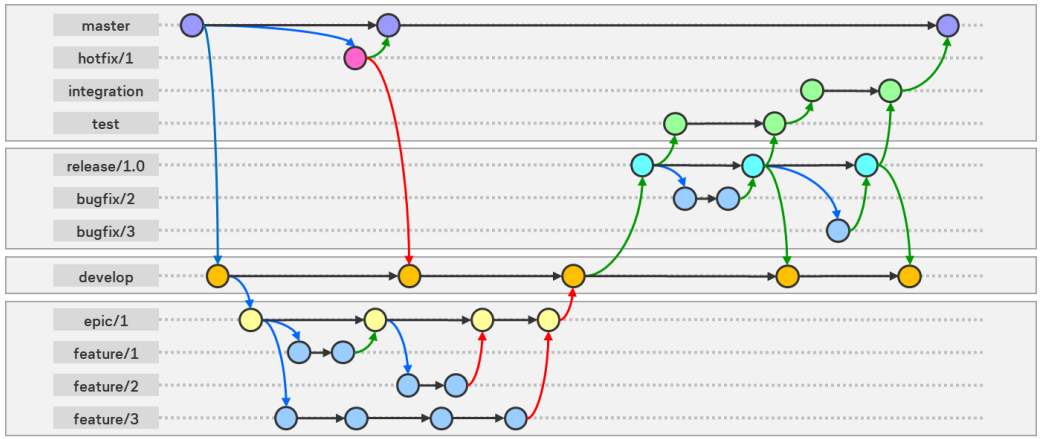
Assessment

✓ Resilience Patterns are used on database

✓ Logging and Monitoring is adequately covered

# Dev Process Branch Management

- Branches used to isolate development tasks
  - Feature-Branches
  - Epics to consolidate Features
- Branches used to coordinate / assemble releases
  - Epic and Release Branches are not always in sync
  - Cherry Picking of changes Epic → Release
- Branches used to promote changes to test stages
  - Release → Test → Integration → Master (=Production)




## Assessment

- ✘ Branching model is very complex
- ➤ Eliminate the Epic-Branches to achieve a more streamlined approach
- ➤ Goal: Reduce necessity of Cherry Picking

# Nach dem Review

- Bestätigung des aktuellen Microservice-Ansatzes
  - Entscheidend zur schrittweisen Ablösung des bestehenden Legacy-Systems über einen Zeitraum von drei Jahren.
- Umgang mit externen Abhängigkeiten
  - Die hohe Abhängigkeit von externen Webservices birgt Risiken, kann jedoch nicht vollständig aufgelöst werden.
  - Diesem Risiko wird durch den Einsatz von Resilienz Mustern begegnet.
  - Diese werden kontinuierlich Weiterentwickelt und ausgebaut, um die Systemstabilität zu erhöhen.
- Logging und Monitoring
  - Nutzung des Elastic Stacks als Muster und Vorbild für zukünftige Entwicklungen

# Best Practices

# Voraussetzungen, damit Reviews nachhaltigen Wert schaffen

## Faire und angemessene Bewertung

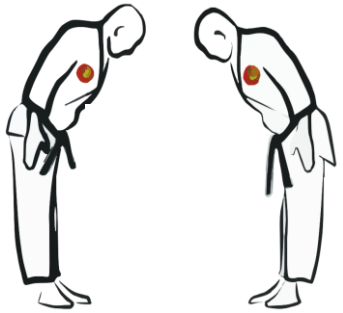
- Ein real-existierendes, komplexes Projekt wird **immer** Befunde haben
- Faire und angemessene Bewertung jenseits oberflächlicher Symptome
  - Wo haben wir wirklich ein Risiko?
  - Wie groß ist der Aufwand zur Behebung – der Anteil an der Technischen Schuld
- Die menschlichen Faktoren erkennen und berücksichtigen

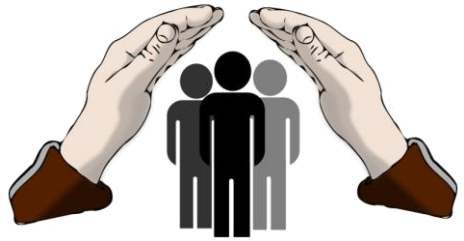
## Ergebnisorientierung

- Wie kann die Situation des Projekts verbessert werden?
- Handlungsempfehlungen zum weiteren Vorgehen – Blick in die Zukunft
- Keine Frage persönlicher Schuld, keine Vergangenheitsbewältigung

# Best Practices: Die “weichen Faktoren”

- Mindset
  - Respekt und Achtsamkeit als Grundlage
  - Neutralität und Professionalität
  - Der Reviewer ist kein Inquisitor!
- Setting: Geschützter Rahmen
  - Fairer Umgang, Transparenz und Integrität
  - Der Rahmen muss **aktiv etabliert** werden
- Offene Kommunikation
  - „Mauern“ gefährdet den Erfolg des Reviews





# Best Practices zur Vorbereitung

- Vorbereitung
  - Klare Zielstellung unerlässlich: Review Auftrag
  - Abstimmung mit Auftraggeber im Vorfeld – später wird die Timebox gefährdet
  - Unterstützung aller Stakeholder sichern: PL und Beteiligte

Phase 1

- Schätzmodell
  - Ausgehend meist von der zweiten Phase:
    Fragestellung auf notwendige Interviews bzw. Deep-Dives abbilden
  - Größe des Systems und Besonderheiten berücksichtigen

# Best Practices zu Durchführung und Aufbereitung

- Durchführung
  - Arbeitsmodus Interview
    - So persönlich wie möglich: In „real-life“ – oder zumindest mit Video-Call
    - Kompakte Termine – ca. 1-1,5h; ggf. Folgetermine
  - Protokolle sind obligatorisch zur Verifikation der Findings
  - Feedback gesteuert: Neue Erkenntnisse berücksichtigen
  - Fokus auf Beobachten und Hinterfragen, um zu verstehen

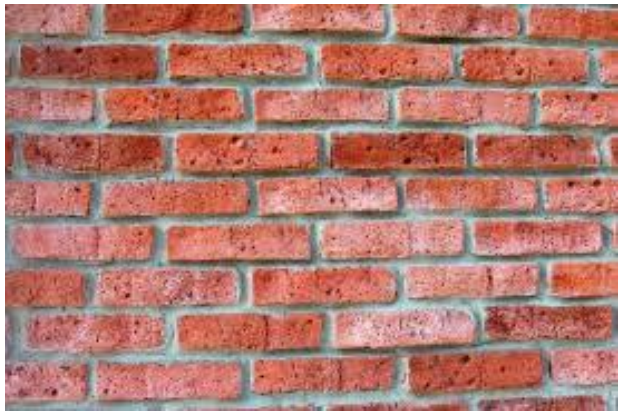
Phase 2

- Aufbereitung
  - Konsolidierung basierend auf den Protokollen der Interviews
  - Informationen gruppieren und kondensieren
  - Fokus auf Bewerten – denn erst jetzt liegt ein komplettes Bild vor.
  - Ergebnispräsentation als Workshop mit den Beteiligten, Gelegenheit zum Dialog

Phase 3

# Die Zerstörung des Reviews
# Pitfalls und Antipattern

# Antipattern: Mauern



- Die Interviewpartner „mauern“
- Risiko
  - Das Review dringt nicht zu den Ursachen der Probleme vor
  - Oberflächliche Formalpunkte stehen im Vordergrund
- Caveat
  - Ohne die Mitarbeit / Offenheit der Beteiligten kann das Review nicht gelingen:
  - *Das Wissen liegt bei den Beteiligten.*

# Antipattern: Wer hat das gesagt?

- Unsicherheit in der Ergebnissicherung
- Wo kommen Aussagen her?
- Information Overload: Sich nicht / Falsch erinnern

- Risiko
  - Nachvollziehbarkeit der Aussagen in Gefahr
  - Fakten gehen verloren oder verfälscht

- Caveat
  - Zu jedem Interview unbedingt ein Protokoll
  - Diese sind mit den Interviewpartnern abzustimmen – erst dann ist das eine „abgesicherte“ Aussage
  - Konsolidierung: Alle Protokolle → Mindmap zum Überblick

# Antipattern: Vorschnelles Urteil

- Nach den ersten Sätzen im Interview „weiß“ der Reviewer schon, was los ist

- Risiko
  - Fehlschluss
  - Die Neutralität ist gefährdet
  - Besondere Aufprägung: Der Reviewer äußert im Interview seine Einschätzung oder solidarisiert sich mit einer Gruppe

- Caveat
  - Klare Trennung: Zuerst beobachten, erst dann bewerten!
    - Beobachten: In Phase 2; Bewerten: In Phase 3
  - Verschiedene Stakeholder haben unterschiedliche Sichten

# Zusammenfassung Architektur-Review

- Komplettes Bild der Architektur gewinnen
  - Einbeziehung aller Quellen, wie Stakeholder, Dokumente und Code
  - Ziel: Tragfähigkeit und Zukunftsfähigkeit der Softwarearchitektur

- Ergebnisorientierter Ansatz, um die Projektsituation zu verbessern
  - Neutrale Sicht von Außen mit Fokus auf Verbesserungspotenziale
  - Konkrete Empfehlungen zum weiteren Vorgehen

- Ausführung in kompakter ca. 6-wöchiger Timebox
  - Ausgeführt von einem Kreis erfahrener Architekten
  - Bewährtes Vorgehensmodell, abgestimmt auf die aktuelle Projektsituation

# Danke für die Aufmerksamkeit

Joachim Nelz
S&N Invent GmbH
Frankfurter Straße 175
Eschborn